Algebraic Reasoning for Probabilistic Action Systems and While-Loops

Larissa Meinicke
Ian J. Hayes

September 2006

Technical Report SSE-2006-05

Division of Systems and Software Engineering Research
School of Information Technology and Electrical Engineering
The University of Queensland
QLD, 4072, Australia

http://www.itee.uq.edu.au/~sse

Algebraic Reasoning for Probabilistic Action Systems and While-Loops

Larissa Meinicke and Ian J. Hayes

School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, Australia
September 4, 2006

Abstract. Back and von Wright have developed algebraic laws for reasoning about loops in the refinement calculus. We extend their work to reasoning about probabilistic loops in the probabilistic refinement calculus. We apply our algebraic reasoning to derive transformation rules for probabilistic action systems and probabilistic while-loops. In particular we focus on developing data refinement rules for these two constructs. Our extension is interesting since some well known transformation rules that are applicable to standard programs are not applicable to probabilistic ones: we identify some of these important differences and we develop alternative rules where possible. In particular, our probabilistic action system and while-loop data refinement rules are new: they differ from the non-probabilistic rules.

1 Introduction

Back and von Wright [5] have used algebraic rules from fixpoint theory to derive transformation rules for loop constructs in the refinement calculus [2, 17]. These transformation rules have been used to reason about practical program derivations, such as data refinement and atomicity refinement of action systems. Such derivations were traditionally verified using either informal or semantic arguments [5]. The algebraic approach has advantages over these methods because it can be used to construct simpler proofs that are easier to check.

The basic algebraic properties of probabilistic programs in the probabilistic refinement calculus [14] differ from those of standard[1] programs. Hence not all the rules for reasoning algebraically about loops developed by Back and von Wright [5] apply in the probabilistic case. The goal of this paper is to develop an equivalent set of rules that are applicable to probabilistic programs. We demonstrate how these rules may be used to generate transformation rules for probabilistic action systems and while-loops. Primarily, we focus on the construction of data refinement rules for probabilistic action systems and while-loops, and we analyse how our data refinement rules may be decomposed into rules that are simpler to use in practice. Our forward simulation rule for probabilistic action

[1] We use the term *standard* to mean nonprobabilistic.

systems is of particular interest, since the forward simulation rule for standard action systems is not applicable in the probabilistic case.

Others (for example McIver and Morgan [18, 14] and Hurd [11]) have demonstrated how to reason formally about probabilistic loops, using invariant based techniques, directly in probabilistic program semantics. Our work on reasoning algebraically about loop transformations may be seen as a complement to theirs, and vice versa.

In the standard refinement calculus [2], the semantics of sequential imperative programs that may include angelic and demonic nondeterminism can be represented using *predicate transformers*. The probabilistic refinement calculus [14] is a generalisation of the refinement calculus, in which programs may also include discrete probabilistic choice. The semantics of probabilistic programs can be modeled using *expectation transformers* [20, 13]. In this paper we use expectation transformers to describe the semantics of probabilistic programs. Our model of expectation transformers differs slightly from that of Morgan and McIver [20, 13], since we have extended their model in order to facilitate the expression of miraculous programs, which is required for modeling guards. Standard programs that may include demonic, but not angelic nondeterminism, are characterised by the *conjunctive* predicate transformers. Likewise the property that characterises probabilistic programs that may include discrete probabilistic choice and demonic, but not angelic nondeterminism, is *sublinearity*$^+$. In their paper, Back and von Wright [5] primarily focused on developing algebraic rules for the conjunctive predicate transformers. In this paper, we mainly focus on developing algebraic transformation rules for sublinear$^+$ expectation transformers. We find that some well known algebraic laws that apply to conjunctive predicate transformers, do not in general apply to sublinear$^+$ expectation transformers. This is mainly because, unlike conjunctive predicate transformers, sublinear$^+$ expectation transformers do not necessarily satisfy right distributivity (R; $(S \sqcap T)$ is in general not equal to R; $S \sqcap R$; T). We identify some of these important rules and supply alternative ones where possible.

The original loop algebra presented by Back and von Wright [5] was a *concrete* algebra, that is it was verified with respect to an underlying semantic model. In later work, von Wright constructed an *abstract* refinement algebra that is independent of a particular program model [24]. Solin and von Wright [22] have further extended this abstract refinement algebra with enabledness and termination axioms and used these to reason about action systems on an abstract level. Their algebra is similar to the Kozen's *Kleene algebra with tests* [12], and Cohen's *Omega algebra* [7], however it differs because it deals with total correctness as well as partial correctness. None of these abstract algebras are sufficient for reasoning about probabilistic programs because they include right distributivity as an axiom. The *lazy Kleene algebra* of Möller [16] is a relaxation of Kleene algebra in which strictness and right-distributivity are omitted. Although Möller's lazy Kleene algebra supports the lack of strictness (R; magic is not in general equal to magic) and right distributivity required for the probabilistic programs presented here, our probabilistic programs do not satisfy Möller's iteration ax-

ioms. In further work on the refinement algebra, von Wright [25] presents axioms of a *general refinement algebra*, which may be verified to be correct with respect to the more general class of *monotonic* predicate transformers, which do not in general satisfy right distributivity. We find that these axioms are valid here for the sublinear$^+$, and more generally the monotonic, expectation transformers. However, few propositions had been derived and verified from these axioms, and we present further interesting properties that could also be included as axioms in the general refinement algebra.

This paper elaborates on our earlier work [15]. It includes a more thorough treatment of how expectation transformers may be extended to include miraculous program behaviour. There are some new transformation rules and there are extended explanations of others. We have extended the action system data refinement rules from [15] to include the treatment of stuttering actions, and we have analysed how the data refinement rules may be decomposed into simpler forms. The probabilistic while-loop transformation rules are also new.

Throughout the paper we refer to results that are new to our approach (or differ from those that apply to conjunctive predicate transformers) as theorems; results which have been proved elsewhere, or whose existing proof applies straightforwardly in our case are referred to as lemmas. Functions are expressed using lambda notation (e.g. $(\lambda\, x \bullet 2 \times x)$), and function application is denoted using an infix dot (e.g. $f.x$). Existential quantifiers are assigned the lowest precedence and we use parentheses to delimit their scope.

In the following section we briefly describe the *expectation transformer* model of Morgan and McIver [14] for probabilistic programs. We extend this model so that miraculous programs can be expressed, and we verify that sublinear expectation transformers are *cocontinuous*. In Sect. 3 the iteration constructs are introduced, and algebraic properties of these constructs are presented and verified. Probabilistic action systems are introduced in Sect. 4, and algebraic rules are constructed to reason about them: in particular we focus on data refinement rules. Our data refinement rules allow for stuttering actions. We demonstrate how our general data refinement rules may be decomposed into simpler forms. In Sect. 5 we then explore the algebraic properties of probabilistic while-loops.

2 Expectation Transformers

Standard (non-probabilistic) imperative programs may be described using a weakest precondition semantics [9]. Similarly, imperative probabilistic programs in which discrete probabilistic choices as well as angelic and demonic nondeterministic choices may be made, may be described using the weakest expectation semantics of McIver and Morgan [20, 13, 14]. We assume that the reader is familiar with such semantics and the basic notions of probabilistic program refinement, as well as the predicate transformer semantics of standard (non-probabilistic programs). We briefly describe the notion of states, expectations and expectation transformers that are used in this paper. We have extended the

Let ϕ and ψ be of type $\mathcal{E}\Sigma$, p and q be of type $\mathcal{P}\Sigma$, c be a constant of type $\mathbb{R}^{\infty}_{\geq 0}$, and c' be of type $\mathbb{R}_{\geq 0}$. When applied to real numbers, $\sqcap$ is the minimum operator (meet), $\sqcup$ is the maximum operator (join), and $\times$ denotes multiplication. We define $0 \times \infty = 0$.

$\mathcal{E}\Sigma$	$\Sigma \to \mathbb{R}^{\infty}_{\geq 0}$	$\mathcal{P}\Sigma$	$\Sigma \to \{0,1\}$
$\phi \leq \psi$	$(\forall \sigma : \Sigma \bullet \phi.\sigma \leq \psi.\sigma)$	$p \Rightarrow q$	$p \leq q$
$\phi \sqcap \psi$	$(\lambda \sigma : \Sigma \bullet \phi.\sigma \sqcap \psi.\sigma)$	$p \wedge p$	$p \sqcap q$
$\phi \sqcup \psi$	$(\lambda \sigma : \Sigma \bullet \phi.\sigma \sqcup \psi.\sigma)$	$p \vee q$	$p \sqcup q$
$\phi \times \psi$	$(\lambda \sigma : \Sigma \bullet \phi.\sigma \times \psi.\sigma)$	True	$(\lambda \sigma : \Sigma \bullet 1)$
$\phi * c$	$(\lambda \sigma : \Sigma \bullet \phi.\sigma \times c)$	False	$(\lambda \sigma : \Sigma \bullet 0)$
$\phi \ominus c'$	$(\lambda \sigma : \Sigma \bullet (\phi.\sigma - c') \sqcup 0)$		
$\neg\phi$	$(\lambda \sigma : \Sigma \bullet 1 - \phi.\sigma)$		

Fig. 1. Expectation notation.

work of McIver and Morgan to deal with miraculous programs so that we may express guards. This extension only results in minor differences in the model.

2.1 Expectation Transformers

In order to simplify our reasoning we assume that we are only dealing with programs over finite state spaces[2].

An *expectation* on a state space Σ, is a function from Σ to $\mathbb{R}^{\infty}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ is the set of positive real numbers and $\mathbb{R}^{\infty}_{\geq 0}$ is defined as $\mathbb{R}_{\geq 0} \cup \{\infty\}$. McIver and Morgan define expectations to be functions from states to the positive real numbers (excluding infinity) [20]. We extend this to include infinity so that we may model guards. (In order to model guards we need to be able to express magical behaviour.) We say that an expectation ϕ of type $\mathcal{E}\Sigma$ is *bounded* if there exists a constant c of type $\mathbb{R}$ such that, for all σ of type Σ, $\phi.\sigma \leq c$.

Fig. 1 formally defines the set of expectations on a given state space Σ, $\mathcal{E}\Sigma$, and operators that are defined on them. Expectations are ordered with respect to the $\leq$ operator. The set of predicates on a state space Σ, $\mathcal{P}\Sigma$, are a subset of expectations: we equate the boolean value true with 1 and false with 0. Given a state space Σ, predicate True is defined as $(\lambda \sigma : \Sigma \bullet 1)$, and False is defined as $(\lambda \sigma : \Sigma \bullet 0)$. For predicates we use the operator $\wedge$ to mean $\sqcap$, $\vee$ to mean $\sqcup$, and $\Rightarrow$ to mean $\leq$. Since the set $\mathbb{R}^{\infty}_{\geq 0}$ forms a complete lattice, by extension, the set of all expectations on any given state space Σ forms a complete lattice, where the top expectation is $(\lambda \sigma : \Sigma \bullet \infty)$, and the least expectation is False.

[2] McIver and Morgan have extended their work on expectation transformer semantics to deal with infinite state spaces [13]. Our finite state space assumption mainly influences our treatment of the healthiness properties. For example, our proof of cocontinuity of sublinear expectation transformers depends on the finite state assumption. Extending our results would require more complex arguments that are outside the scope of this paper.

Expectation transformers are used to model probabilistic programs [14]. An expectation transformer is a function from expectations on the output state space, Γ, to expectations on the input state space, Σ. Expectation transformers are the probabilistic equivalent of predicate transformers in the refinement calculus [2]: given a predicate transformer S, a predicate p, and an initial state σ, $S.p.\sigma$ is true if S is guaranteed to terminate in a state satisfying p from σ, and false otherwise. Given an expectation transformer S, a predicate p, and an initial state σ, $S.p.\sigma$ specifies the least probability that S will terminate in a state satisfying p from σ. More generally, for any expectation ϕ, $S.\phi.\sigma$ returns the least expectation of ϕ in program S from σ. For example, $S.(\lambda\,\sigma \bullet \sigma.x).\sigma$ returns the least average value of variable x produced by S from σ. Refinement between two expectation transformers $S, T : \mathcal{E}\Gamma \to \mathcal{E}\Sigma$ is defined as

$$S \sqsubseteq T \triangleq (\forall\, \phi : \mathcal{E}\Gamma \bullet S.\phi \leq T.\phi).$$

By extension, the set of all expectation transformers forms a complete lattice, where the top element is magic, and the least element is abort (see Fig. 2). We say that an expectation transformer S of type $\mathcal{E}\Gamma \to \mathcal{E}\Sigma$ is *miraculous* from a state σ of type Σ if

$$(\forall\, \phi : \mathcal{E}\Gamma \bullet S.\phi.\sigma = \infty).$$

Basic Operators and Commands. The basic operators and commands are shown in Fig. 2. For predicate g, the assertion command $\{g\}$ aborts from states in which g does not hold and performs no action from states satisfying g, while the guard $[g]$ is miraculous from states which do not satisfy g, and performs no action from other states. The assertion command is also defined more generally when g is an expectation: this extended definition of an assertion is novel.

The four basic composition operators are probabilistic, demonic choice, angelic choice, and sequential composition. The iteration operators, which are discussed in Sect. 3, have the highest precedence, followed by sequential composition, and then with equal precedence demonic, angelic, and probabilistic choice. From initial state σ, the probabilistic choice statement $S_p\oplus S'$, performs S with probability $p.\sigma$, and S' with probability $1 - p.\sigma$. The demonic choice operator on expectation transformers is defined using the meet (pointwise minimum) operator, while the angelic choice operator is defined using the join (pointwise maximum) operator. Given the definition of refinement between expectation transformers, a demonic choice, $S \sqcap S'$, is able to be refined by any probabilistic choice between S and S'. The unit of demonic choice is magic, the unit of angelic choice is abort, and the unit of sequential composition is skip, the program that does not modify the state.

Note that in our model, for any expectation transformer S, we have specified that program $S_p\oplus$magic is miraculous for states σ from which $p.\sigma$ does not equal one. It is not our intention to be able to express programs that are miraculous with some probability between 0 and 1. This may seem to be restrictive, and there are other ways to model miraculous behaviour in probabilistic programs in which we are able to distinguish between programs that are miraculous with certain probabilities [19]. However, for the purpose of reasoning about action

Let g be a predicate on state space Σ; θ and ϕ be expectations of type $\mathcal{E}\Sigma$; ψ be an expectation of type $\mathcal{E}\Gamma$; S and S' be expectation transformers of type $\mathcal{E}\Gamma \to \mathcal{E}\Sigma$; $R : \mathcal{E}\Theta \to \mathcal{E}\Sigma$; $R' : \mathcal{E}\Gamma \to \mathcal{E}\Theta$; $T : \mathcal{E}\Sigma \to \mathcal{E}\Sigma$; and p a probability function of type $\Sigma \to [0..1]$, where $[0..1]$ is the closed interval from 0 to 1.

assertion	$\{\theta\}.\phi$	$(\lambda\, \sigma : \Sigma \bullet \theta.\sigma \times \phi.\sigma)$
guard	$[g].\phi$	$(\lambda\, \sigma : \Sigma \bullet \mathsf{if}\ g.\sigma\ \mathsf{then}\ \phi.\sigma\ \mathsf{else}\ \infty)$
bottom	$\mathsf{abort}.\phi$	False
top	$\mathsf{magic}.\phi$	$(\lambda\, \sigma : \Sigma \bullet \infty)$
unit of composition	$\mathsf{skip}.\phi$	ϕ
probabilistic choice	$(S_p \oplus S').\psi$	$(\{p\};\ S).\psi + (\{\neg p\};\ S').\psi$
demonic choice	$(S \sqcap S').\psi$	$S.\psi \sqcap S'.\psi$
angelic choice	$(S \sqcup S').\psi$	$S.\psi \sqcup S'.\psi$
sequential composition	$(R;\ R').\psi$	$R.(R'.\psi)$
strong iteration	$(T^\omega).\phi$	$(\mu\, X \bullet T;\ X \sqcap \mathsf{skip}).\phi$
weak iteration	$(T^*).\phi$	$(\nu X \bullet T;\ X \sqcap \mathsf{skip}).\phi$
infinite iteration	$(T^\infty).\phi$	$(\mu\, X \bullet T;\ X).\phi$

Fig. 2. Weakest expectation semantics for probabilistic operators.

systems and while-loops, we find that we do not need such a richer semantics. Care must be taken when using alternative models, since they are unlikely to share all of the same properties as our current model.

When predicates are used, guards and assertions satisfy many of the same basic properties as they satisfy in the standard refinement calculus [5]. For predicates p and q,

$$
\begin{array}{lcl@{\qquad}lcl}
[p] \sqsubseteq [q] & \Leftrightarrow & q \le p, & \{p\} \sqsubseteq \{q\} & \Leftrightarrow & p \le q, \\
[p];\ [q] & = & [p \wedge q], & \{p\};\ \{q\} & = & \{p \wedge q\}, \\
[p] \sqcap [q] & = & [p \vee q], & \{p\} \sqcap \{q\} & = & \{p \wedge q\}, \\
[p] \sqcup [q] & = & [p \wedge q], & \{p\} \sqcup \{q\} & = & \{p \vee q\}, \\
[p] & = & [p];\ \{p\}, & \{p\} & = & \{p\};\ [p], \\
\mathsf{skip} & \sqsubseteq & [p], & \{p\} & \sqsubseteq & \mathsf{skip}, \\
\{p\} & = & [\neg p];\ \mathsf{abort} \sqcap [p], \text{ and} & [p] & = & \{\neg p\};\ \mathsf{magic} \sqcup \{p\}.
\end{array}
$$

Healthiness Properties. As for predicate transformers, expectation transformers can be classified by a number of *healthiness* properties [14] (Fig. 3). Primarily we consider *sublinear*, *infinitely scalable* expectation transformers, which we refer to as the *sublinear*$^+$ expectation transformers.

Standard (non-probabilistic) programs with demonic choice, but no angelic choice, are characterised by the set of *conjunctive* predicate transformers. Similarly, in Morgan and McIver's model of expectation transformers [20], the *sublinear*[3] expectation transformers characterise the set of probabilistic programs that may be expressed using the relational probabilistic model of He et al. [10]:

[3] Our definition of sublinearity differs from that of Morgan and McIver [20] because we allow the scaling constants c_1 and c_2 to be infinite.

Let S be an expectation transformer of type $\mathcal{E}\Gamma \to \mathcal{E}\Sigma$, c_1 and c_2 be constants of type $\mathbb{R}^{\infty}_{\geq 0}$, c be a constant of type $\mathbb{R}_{\geq 0}$, β_1 and β_2 be expectations of type $\mathcal{E}\Gamma$, $\mathcal{B}$ and $\mathcal{B}'$ be directed and codirected sets respectively, of expectations of type $\mathcal{E}\Gamma$.

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

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

$$\beta_1 \leq \beta_2 \Rightarrow S.\beta_1 \leq S.\beta_2 \qquad \text{(monotonicity)}$$

$$(\forall\, \sigma : \Sigma, \phi : \mathcal{E}\Gamma \bullet S.\phi.\sigma \leq (\sqcap \gamma \bullet \phi.\gamma)) \qquad \text{(feasibility)}$$

$$(\forall\, \sigma : \Sigma \bullet (\forall\, \phi : \mathcal{E}\Gamma \bullet S.\phi.\sigma \leq (\sqcap \gamma \bullet \phi.\gamma)) \vee (\forall\, \phi : \mathcal{E}\Gamma \bullet S.\phi.\sigma = \infty)) \qquad \text{(01-feasibility)}$$

$$S.(c * \beta) = c * S.\beta \qquad \text{(finite scaling)}$$

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

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

$$S.(\sqcap \beta : \mathcal{B}' \bullet \beta) = (\sqcap \beta : \mathcal{B}' \bullet S.\beta) \qquad \text{(cocontinuity)}$$

Fig. 3. Healthiness properties for expectation transformers.

a model that captures discrete probabilistic and demonic nondeterministic behaviour, but not angelic nondeterministic behaviour. In our model the sublinear$^+$ expectation transformers are interesting because they characterise a slight variation in the model of He et al.: a variation in which magical behaviour may also be expressed[4]. The operators given in Fig. 2, apart from angelic choice, preserve sublinearity$^+$ of their arguments.

Not all sublinear$^+$ expectation transformers are conjunctive. However, expectation transformers that may be written in terms of the operators and commands other than probabilistic choice, angelic choice, and assertions expressed using expectations other than predicates, are. Morgan and McIver verified that, given their definitions of sublinearity and expectation transformers, sublinear expectation transformers are *monotonic* [20]. This is true for our extended model as well. An expectation transformer S of type $\mathcal{E}\Gamma \to \mathcal{E}\Sigma$ is *feasible* from a given initial state σ of type Σ, if

$$(\forall\, \phi : \mathcal{E}\Gamma \bullet S.\phi.\sigma \leq (\sqcap \gamma : \Gamma \bullet \phi.\gamma)).$$

An expectation transformer that is feasible from every initial state is called *feasible* [20]. For their expectation model, McIver and Morgan [20] showed that the

[4] The original semantics of He et. al. did not facilitate the expression of magical behaviour: programs were expressed as a function from input states to non-empty sets of discrete distributions over the output states that satisfy certain closure properties (see [20, 14] for more details). In order to express magical behaviour, we simply remove the assumption that the sets of distributions must be non-empty. For our minor extension, the correspondence proof used by McIver and Morgan applies with only slight modifications

sublinear expectation are feasible: here, since we are able to express miraculous programs, this is not the case. However, our sublinear expectation transformers do satisfy the *01-feasibility* property. This property states that, from any initial state σ, an expectation transformer S is either feasible, or it is miraculous.

Lemma 1. *For any sublinear expectation transformer S, S satisfies the* 01-feasibility *property.*

Proof. Let S be of type $\mathcal{E}\Gamma \to \mathcal{E}\Sigma$. First we verify that, for all σ of type Σ, $S.\mathsf{False}.\sigma \in \{0, \infty\}$.

$$
\begin{array}{ll}
 & S.\mathsf{False}.\sigma \\
= & \{\text{definition of } \mathsf{False} \text{ and scalar multiplication }\} \\
 & S.(2 * \mathsf{False}).\sigma \\
\geq & \{\text{sublinearity}\} \\
 & (2 * S.\mathsf{False}).\sigma \\
= & \{\text{definition of scalar multiplication}\} \\
 & 2 \times S.\mathsf{False}.\sigma
\end{array}
$$

This inequality only holds for values of $S.\mathsf{False}.\sigma$ that are either 0 or ∞. We now consider both of these cases in turn. If, for a given σ, we have that $S.\mathsf{False}.\sigma = 0$, then we are required to show that S is feasible from σ. That is, we are required to show that for all ϕ of type $\mathcal{E}\Gamma$, $S.\phi.\sigma \leq (\sqcap \gamma : \Gamma \bullet \phi.\gamma)$. If ϕ is bounded above then we have that

$$
\begin{array}{ll}
 & 0 \\
= & \{\text{assumption}\} \\
 & S.\mathsf{False}.\sigma \\
= & \{\phi \text{ is bounded}\} \\
 & S.(\phi \ominus (\sqcap \gamma : \Gamma \bullet \phi.\gamma)).\sigma \\
\geq & \{\text{sublinearity}\} \\
 & S.\phi.\sigma \ominus (\sqcap \gamma : \Gamma \bullet \phi.\gamma),
\end{array}
$$

and so $S.\phi.\sigma \leq (\sqcap \gamma : \Gamma \bullet \phi.\gamma)$. If ϕ is not bounded above, then $(\sqcap \gamma : \Gamma \bullet \phi.\gamma) = \infty$, and our inequality trivially holds. For the case where $S.\mathsf{False}.\sigma = \infty$, from monotonicity we have that for all ϕ of type $\mathcal{E}\Gamma$, $S.\phi.\sigma = \infty$, and so S is magical from σ. □

With respect to their definitions, McIver and Morgan [20] also proved that the sublinear expectation transformers are *finitely scalable*. In our model, for non-zero constant c, this may be verified to be true in the same way. However, for scaling constant $c = 0$, this does not hold for infeasible expectation transformers. Given a sublinear, infeasible expectation transformer S of type $\mathcal{E}\Gamma \to \mathcal{E}\Sigma$, from the 01-feasibility property we have that there exists a σ of type Σ such that S is miraculous from σ. Given such a state σ we have that,

$$
\begin{array}{ll}
& S.(0 * \phi).\sigma \\
= & \{\text{definition of scalar multiplication}\} \\
& S.\mathsf{False}.\sigma \\
= & \{\text{assumption } S \text{ is miraculous from } \sigma\} \\
& \infty \\
\neq & \{0 \times \infty = 0\} \\
& 0 \times S.\phi.\sigma \\
= & \{\text{definition of scalar multiplication}\} \\
& (0 * S.\phi).\sigma.
\end{array}
$$

Hence, $S.(0 * \phi) \neq 0 * S.\phi$. However, scaling by 0 trivially hold for feasible programs: for feasible expectation transformer S,

$$
\begin{array}{ll}
& S.(0 * \phi) \\
= & \{\text{definition of scalar multiplication}\} \\
& S.\mathsf{False} \\
= & \{\text{assumption } S \text{ is feasible}\} \\
& \mathsf{False} \\
= & \{\text{definition of scalar multiplication}\} \\
& 0 * \mathsf{False}.
\end{array}
$$

Continuity and cocontinuity are important properties because they simplify the treatment of least and greatest fixpoints over a complete lattice. Their definition involves the use of directed, and codirected sets, which are defined as follows [8][5]. For any subset $\mathcal{B}$ of a partially ordered set $\mathcal{A}$,

$$
\begin{array}{r}
\mathit{directed}.\mathcal{B} \triangleq \mathcal{B} \neq \{\} \wedge (\forall\, \alpha, \beta : \mathcal{B} \bullet (\exists\, \gamma : \mathcal{B} \bullet \alpha \sqsubseteq \gamma \wedge \beta \sqsubseteq \gamma)) \text{ and} \\
\mathit{codirected}.\mathcal{B} \triangleq \mathcal{B} \neq \{\} \wedge (\forall\, \alpha, \beta : \mathcal{B} \bullet (\exists\, \gamma : \mathcal{B} \bullet \gamma \sqsubseteq \alpha \wedge \gamma \sqsubseteq \beta)).
\end{array}
$$

A codirected set is the dual of a directed set, and cocontinuity is the dual of continuity. Continuity and cocontinuity may be more generally defined for any monotonic function over complete partial orders. For monotonic function $f : A \to B$, where A and B are complete partial orders, and directed set $\mathcal{X}$ of A and codirected set $\mathcal{X}'$ of A, f is continuous if

$$f.(\sqcup X : \mathcal{X} \bullet X) = (\sqcup X : \mathcal{X} \bullet f.X),$$

and it is cocontinuous if

$$f.(\sqcap X : \mathcal{X}' \bullet X) = (\sqcap X : \mathcal{X}' \bullet f.X).$$

Next we verify that sublinear expectation transformers are *cocontinuous*. Previously, this property had not been verified to be correct with respect to the expectation model of McIver and Morgan [20]: our proof applies to their expectation transformer model as well. Our proof of cocontinuity for expectation transformers is similar to McIver and Morgan's proof of bounded continuity [14].

Theorem 2 (cocontinuity). *Sublinear expectation transformers are cocontinuous.*

[5] This definition differs from that used by Back and von Wright [2] since it constrains directed and codirected sets to be non-empty.

Proof. For any sublinear expectation transformer $T : \mathcal{E}\Sigma \to \mathcal{E}\Gamma$, and $\mathcal{B}$ a $\leq$-codirected subset of $\mathcal{E}\Sigma$, we are required to show that

$$T.(\sqcap\beta : \mathcal{B} \bullet \beta) = (\sqcap\beta : \mathcal{B} \bullet T.\beta).$$

By monotonicity of expectation transformers we only need to show that $T.(\sqcap\beta : \mathcal{B} \bullet \beta) \geq (\sqcap\beta : \mathcal{B} \bullet T.\beta)$.

For any constant $c > 0$, for each state $\sigma : \Sigma$, there exists an expectation $\beta_\sigma : \mathcal{B}$ such that $\beta_\sigma.\sigma \ominus c \leq (\sqcap\beta : \mathcal{B} \bullet \beta).\sigma$. Since $\mathcal{B}$ is codirected, and the state space Σ is finite we then have that there exists a $\beta_c : \mathcal{B}$ such that for all $\sigma : \Sigma$, $\beta_c.\sigma \leq \beta_\sigma.\sigma$, and hence $\beta_c \ominus c \leq (\sqcap\beta : \mathcal{B} \bullet \beta)$. We then have that

$$\begin{array}{ll} & T.(\sqcap\beta : \mathcal{B} \bullet \beta) \\ \geq & \{\text{monotonicity and the above}\} \\ & T.(\beta_c \ominus c) \\ \geq & \{\text{sublinearity}\} \\ & T.\beta_c \ominus c \\ \geq & \{\beta_c \in \mathcal{B} \text{ and monotonicity}\} \\ & (\sqcap\beta : \mathcal{B} \bullet T.\beta) \ominus c, \end{array}$$

which suffices because c may be arbitrarily close to zero. □

Continuity of an expectation transformer R, guarantees continuity of function $f = (\lambda\, X \bullet R;\ X)$, and similarly, cocontinuity of R, guarantees cocontinuity of f. We verify this for cocontinuity, the proof for continuity is similar.

Theorem 3. *For any cocontinuous expectation transformer R, the function $f = (\lambda\, X \bullet R;\ X)$ is cocontinuous.*

Proof. Let $\mathcal{X}$ be a $\sqsubseteq$-codirected set of expectation transformers. From the definition of refinement it is sufficient to show that for all ϕ

$$\begin{array}{ll} & (R;\ (\sqcap X : \mathcal{X} \bullet X)).\phi \\ = & \{\text{definition of sequential composition}\} \\ & R.((\sqcap X : \mathcal{X} \bullet X).\phi) \\ = & \{\ \sqcap \text{ defined pointwise over expectation transformers}\} \\ & R.(\sqcap X : \mathcal{X} \bullet X.\phi) \\ = & \{\text{cocontinuity of } R \text{ (since } \mathcal{X} \text{ is codirected, the set} \\ & \ \{X : \mathcal{X} \bullet X.\phi\} \text{ is codirected, see below for justification)}\} \\ & (\sqcap X : \mathcal{X} \bullet R.(X.\phi)) \\ = & \{\text{definition of sequential composition}\} \\ & (\sqcap X : \mathcal{X} \bullet (R;\ X).\phi) \\ = & \{\ \sqcap \text{ defined pointwise over expectation transformers}\} \\ & (\sqcap X : \mathcal{X} \bullet R;\ X).\phi. \end{array}$$

For all ϕ we have that the set $\mathcal{B} = \{X : \mathcal{X} \bullet X.\phi\}$ is codirected, since

$\mathcal{B} \neq \{\} \wedge (\forall\, \alpha, \beta : \mathcal{B} \bullet (\exists\, \gamma : \mathcal{B} \bullet \gamma \leq \alpha \wedge \gamma \leq \beta))$
$\Leftrightarrow$ {definition of $\mathcal{B}$}
$\mathcal{X} \neq \{\} \wedge (\forall\, X, Y : \mathcal{X} \bullet (\exists\, Z : \mathcal{X} \bullet Z.\phi \leq X.\phi \wedge Z.\phi \leq Y.\phi))$
$\Leftrightarrow$ {$\mathcal{X}$ is codirected, refinement order on expectation transformers}
true.

□

Basic Algebraic Properties. Monotonic expectation transformers share the same set of basic algebraic rules (Fig. 4) as monotonic predicate transformers [24]. Unlike conjunctive predicate transformers, sublinear$^+$ expectation transformers satisfy right sub-distributivity $R;\ (S \sqcap T) \sqsubseteq R;\ S \sqcap R;\ T$, but do not in general satisfy right distributivity $R;\ (S \sqcap T) = R;\ S \sqcap R;\ T$. For example, given

$R \triangleq (x := 0_{\frac{1}{2}} \oplus x := 1)$,
$S \triangleq [x = 0];\ y := 0 \sqcap [x = 1];\ y := 1$, and
$T \triangleq [x = 0];\ y := 1 \sqcap [x = 1];\ y := 0$,

we have that,

$R;\ (S \sqcap T) = (x := 0_{\frac{1}{2}} \oplus x := 1);\ (y := 0 \sqcap y := 1)$, but
$R;\ S \sqcap R;\ T = (x, y := 0, 0_{\frac{1}{2}} \oplus x, y := 1, 1) \sqcap (x, y := 0, 1_{\frac{1}{2}} \oplus x, y := 1, 0)$.

In program $R;\ S \sqcap R;\ T$ we have that y is chosen to be 0 with probability $\frac{1}{2}$, and 1 with probability $\frac{1}{2}$, whereas in $R;\ (S \sqcap T)$, y is not guaranteed to be assigned 0 with probability $\frac{1}{2}$, nor is it guaranteed to be assigned 1 with probability $\frac{1}{2}$: the value of y may be chosen nondeterministically. Hence $R;\ (S \sqcap T)$ is refined by $R;\ S \sqcap R;\ T$, but is not equivalent to it.

$R;\ (S;\ T) = (R;\ S);\ T$	(associativity)
$\mathsf{skip};\ S = S = S;\ \mathsf{skip}$	(unit)
$R \sqcap (S \sqcap T) = (R \sqcap S) \sqcap T$	(associativity)
$\mathsf{magic} \sqcap S = S$	(unit)
$R \sqcap S = S \sqcap R$	(commutativity)
$R \sqcap R = R$	(idempotence)
$(R \sqcap S);\ T = R;\ T \sqcap S;\ T$	(left distributivity)
$R;\ (S \sqcap T) \sqsubseteq R;\ S \sqcap R;\ T$	(right sub-distributivity)
$\mathsf{magic};\ R = \mathsf{magic}$	(preemption)
$\mathsf{abort};\ R = \mathsf{abort}$	(preemption)

Fig. 4. Basic algebraic properties of monotonic expectation transformers.

3 Iteration Constructs

We use the same iteration constructs for probabilistic programs as those that are used for standard programs [5, 2]. These constructs are expressed using fixpoints, and may be reasoned about using the usual fixpoint theory [8, 5, 2].

Lemma 4 (Knaster-Tarski). *Every monotonic function on a complete lattice has a complete lattice of fixpoints.*

Recall from earlier that, because we have introduced the ability to express miraculous behaviour, the set of expectation transformers forms a complete lattice. The least, μ, and greatest, ν, fixpoint operators satisfy the following induction and unfolding properties.

$$f.(\mu\, .f) = \mu\, .f \text{ and } f.(\nu .f) = \nu .f \qquad \text{(unfolding)}$$

$$f.x \sqsubseteq x \Rightarrow \mu\, .f \sqsubseteq x \text{ and } x \sqsubseteq f.x \Rightarrow x \sqsubseteq \nu .f \qquad \text{(induction)}$$

We also use the *rolling rules* for fixpoints [5].

Lemma 5 (rolling). *Given monotonic functions f and g on a complete lattice,*

$$f.(\mu\, .(g \circ f)) = \mu\, .(f \circ g) \text{ and } f.(\nu .(g \circ f)) = \nu .(f \circ g).$$

The following *fusion* lemma [2] (attributed to Kleene) is used to verify a commutativity property (Theorem 21).

Lemma 6 (fusion). *Let $f : \Sigma \to \Sigma$ and $g : \Gamma \to \Gamma$ be monotonic functions on complete lattices Σ and Γ.*

(a) If $h : \Sigma \to \Gamma$ is continuous, then
$h \circ f \sqsubseteq g \circ h \Rightarrow h.(\mu\, .f) \sqsubseteq \mu\, .g$ *and*
$h \circ f = g \circ h \Rightarrow h.(\mu\, .f) = \mu\, .g$.

(b) If $h : \Sigma \to \Gamma$ is cocontinuous, then
$h \circ f \sqsubseteq g \circ h \Rightarrow h.(\nu .f) \sqsubseteq \nu .g$ *and*
$h \circ f = g \circ h \Rightarrow h.(\nu .f) = \nu .g$.

For continuous and cocontinuous functions on a complete lattice, reasoning about least and greatest fixpoints respectively can be reduced to reasoning about finite iterations of the functions [8], where $f^0.x \triangleq x$ and for $i \in \mathbb{N}$ $f^{i+1} = f.(f^i.x)$.

Lemma 7.

(a) The least fixed point of the continuous function f on a complete lattice is the limit

$$\mu\, .f = (\sqcup i : \mathbb{N} \bullet f^i.\bot),$$

where $\bot$ is the bottom element of the complete lattice.

(b) The greatest fixed point of a cocontinuous function, f, on a complete lattice is the colimit

$$\nu .f = (\sqcap i : \mathbb{N} \bullet f^i.\top),$$

where $\top$ is the top element of the complete lattice.

3.1 Iteration Operators

The iteration operators are defined in Fig. 2. Informally, T^* executes T any finite number of times, T^∞ executes T an infinite number of times, and T^ω executes T any infinite or finite number of times. From the definition of our iteration operators, and the induction and unfolding properties of fixpoints we immediately get the usual unfolding and induction rules:

$$R^\omega = R;\ R^\omega \sqcap \mathsf{skip} \qquad \text{(unfold strong iteration)}$$
$$R^* = R;\ R^* \sqcap \mathsf{skip} \qquad \text{(unfold weak iteration)}$$
$$R^\infty = R;\ R^\infty \qquad \text{(unfold infinite iteration)}$$
$$R;\ X \sqcap \mathsf{skip} \sqsubseteq X \Rightarrow R^\omega \sqsubseteq X \qquad \text{(strong iteration induction)}$$
$$X \sqsubseteq R;\ X \sqcap \mathsf{skip} \Rightarrow X \sqsubseteq R^* \qquad \text{(weak iteration induction)}$$
$$R;\ X \sqsubseteq X \Rightarrow R^\infty \sqsubseteq X \qquad \text{(infinite iteration induction)}$$

For a conjunctive expectation transformer R, R^* satisfies the Kleene axioms of Kozen [12] and Cohen [6], in particular,

$$R^* = (\sqcap i : \mathbb{N} \bullet R^i),$$

where for expectation transformer R, $R^0 \triangleq \mathsf{skip}$, and for $i \in \mathbb{N}$, $R^{i+1} = R;\ R^i$. However, this equivalence does not hold in general for sublinear expectation transformers. Instead the definition of *weak* iteration satisfies the following alternative theorem.

Theorem 8 (Kleene star equivalence). *Let R be a monotonic, cocontinuous expectation transformer, then*

$$R^* = (\sqcap i : \mathbb{N} \bullet (R \sqcap \mathsf{skip})^i).$$

Proof. Before verifying our statement we prove four lemmas. The first lemma is used in the verification of lemmas two and three, and the second and third lemmas are used to verify the fourth.

1. $(\forall\, i : \mathbb{N} \bullet R;\ (R \sqcap \mathsf{skip})^i \sqcap \mathsf{skip} = (R \sqcap \mathsf{skip})^{i+1})$
   (a) Base case: $i = 0$
   $R;\ (R \sqcap \mathsf{skip})^0 \sqcap \mathsf{skip}$
   $=$ {definition}
   $R;\ \mathsf{skip} \sqcap \mathsf{skip}$
   $=$ {skip is unit}
   $(R \sqcap \mathsf{skip})^1$
   (b) Inductive case: assume $R;\ (R \sqcap \mathsf{skip})^i \sqcap \mathsf{skip} = (R \sqcap \mathsf{skip})^{i+1}$
   $(R \sqcap \mathsf{skip})^{i+2}$
   $=$ {definition}
   $(R \sqcap \mathsf{skip});\ (R \sqcap \mathsf{skip})^{i+1}$
   $=$ {left distributivity}
   $R;\ (R \sqcap \mathsf{skip})^{i+1} \sqcap \mathsf{skip};\ (R \sqcap \mathsf{skip})^{i+1}$
   $=$ {inductive assumption on second choice}
   $R;\ (R \sqcap \mathsf{skip})^{i+1} \sqcap R;\ (R \sqcap \mathsf{skip})^i \sqcap \mathsf{skip}$
   $=$ {by monotonicity, $(R \sqcap \mathsf{skip})^{i+1} = (R \sqcap \mathsf{skip});\ (R \sqcap \mathsf{skip})^i \sqsubseteq (R \sqcap \mathsf{skip})^i$
   and from basic lattice properties we have that $x \sqsubseteq y \Rightarrow x \sqcap y = x$}
   $R;\ (R \sqcap \mathsf{skip})^{i+1} \sqcap \mathsf{skip}$

2. $(\forall\, i : \mathbb{N} \bullet (\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^{i+1}.\mathsf{magic} \sqsubseteq (R \sqcap \mathsf{skip})^i)$

   (a) Base case: $i = 0$
   $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^{0+1}.\mathsf{magic}$
   $=$ {function application}
   $R;\ \mathsf{magic} \sqcap \mathsf{skip}$
   $\sqsubseteq$ {general lattice rule $x \sqcap y \sqsubseteq x$}
   skip
   $=$ {definition}
   $(R \sqcap \mathsf{skip})^0$

   (b) Inductive case: assume $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^{i+1}.\mathsf{magic} \sqsubseteq (R \sqcap \mathsf{skip})^i$
   $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^{i+2}.\mathsf{magic}$
   $=$ {definition}
   $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip}).((\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^{i+1}.\mathsf{magic})$
   $\sqsubseteq$ {inductive assumption and monotonicity}
   $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip}).(R \sqcap \mathsf{skip})^i$
   $=$ {function application}
   $R;\ (R \sqcap \mathsf{skip})^i \sqcap \mathsf{skip}$
   $=$ {By 1.}
   $(R \sqcap \mathsf{skip})^{i+1}$

3. $(\forall\, i : \mathbb{N} \bullet (R \sqcap \mathsf{skip})^i \sqsubseteq (\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^i.\mathsf{magic})$

   (a) Base case: $i = 0$
   $(R \sqcap \mathsf{skip})^0$
   $\sqsubseteq$ {magic is top element}
   magic
   $=$ {definition}
   $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^0.\mathsf{magic}$

   (b) Inductive case: assume $(R \sqcap \mathsf{skip})^i \sqsubseteq (\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^i.\mathsf{magic}$
   $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^{i+1}.\mathsf{magic}$
   $=$ {definition}
   $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip}).((\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^i.\mathsf{magic})$
   $\sqsupseteq$ {inductive assumption and monotonicity}
   $(\lambda\, X \bullet R;\ X \sqcap \mathsf{skip}).(R \sqcap \mathsf{skip})^i$
   $=$ {function application}
   $R;\ (R \sqcap \mathsf{skip})^i \sqcap \mathsf{skip}$
   $=$ {By 1.}
   $(R \sqcap \mathsf{skip})^{i+1}$

4. $(\sqcap i : \mathbb{N} \bullet (\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^i.\mathsf{magic}) = (\sqcap i : \mathbb{N} \bullet (R \sqcap \mathsf{skip})^i)$

   First we have that,

   $(\sqcap i : \mathbb{N} \bullet (\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^i.\mathsf{magic})$
   $=$ {for any monotonic function f we have that
   $(\sqcap i : \mathbb{N} \bullet f^i.\top = \sqcap i : \mathbb{N} \bullet f^{i+1}.\top)$}
   $(\sqcap i : \mathbb{N} \bullet (\lambda\, X \bullet R;\ X \sqcap \mathsf{skip})^{i+1}.\mathsf{magic})$
   $\sqsubseteq$ {from 2}
   $(\sqcap i : \mathbb{N} \bullet (R \sqcap \mathsf{skip})^i)$.

   The other direction follows from 3.

As a result, the following derivation proves our goal:

$$
\begin{array}{ll}
& R^* \\
= & \{\text{definition}\} \\
& (\nu X \bullet R;\ X \sqcap \mathsf{skip}) \\
= & \{\text{cocontinuity of } (\lambda X \bullet R;\ X \sqcap \mathsf{skip}) \text{ (see below), and Lemma 7}\} \\
& (\sqcap i : \mathbb{N} \bullet (\lambda X \bullet R;\ X \sqcap \mathsf{skip})^i.\mathsf{magic}) \\
= & \{\text{By 4.}\} \\
& (\sqcap i : \mathbb{N} \bullet (R \sqcap \mathsf{skip})^i).
\end{array}
$$

We may verify that function $(\lambda X \bullet R;\ X \sqcap \mathsf{skip})$ is cocontinuous as follows. Let $\mathcal{X}$ be a $\sqsubseteq$-codirected set of expectation transformers. We have that

$$
\begin{array}{ll}
& (\sqcap X : \mathcal{X} \bullet R;\ X \sqcap \mathsf{skip}) \\
= & \{\text{lattice property } (\sqcap x : X \bullet x \sqcap y) = (\sqcap x : X \bullet x) \sqcap y\} \\
& (\sqcap X : \mathcal{X} \bullet R;\ X) \sqcap \mathsf{skip} \\
= & \{\text{cocontinuity of } R \text{ and Theorem 3}\} \\
& R;\ (\sqcap X : \mathcal{X}) \sqcap \mathsf{skip}.
\end{array}
$$

□

Note that for conjunctive R, $(\sqcap i : \mathbb{N} \bullet R^i) = (\sqcap i : \mathbb{N} \bullet (R \sqcap \mathsf{skip})^i)$. For conjunctive predicate transformers we can decompose R^ω into its terminating (R^*) and nonterminating (R^∞) behaviours: that is we have that

$$R^\omega = R^* \sqcap R^\infty.$$

For sublinear expectation transformers this is, in general, not the case. The main reason for this difference is that, from a particular initial state, a standard program may either exhibit non-terminating behaviour (that is it may abort) or it may behave miraculously, or it may terminate in a set of states. A probabilistic program may exhibit some probabilistic distribution of these behaviours: for example it may not terminate (abort) with probability a half, and it may produce some distribution of states with the other half. Because of this, we cannot trivially separate out the strong iteration operator into its finite and infinite behaviours. For example, take $R = [x = 1] \sqcap [x = 0];\ (x := 1 \oplus_{\frac{1}{2}} x := 2)$. We have that

$$
\begin{array}{l}
R^\omega = [x = 1];\ \mathsf{abort} \sqcap [x = 0];\ (\mathsf{abort} \oplus_{\frac{1}{2}} x := 2) \sqcap \mathsf{skip}, \\
R^* = [x = 1];\ \mathsf{skip} \sqcap ([x = 0];\ (x := 1 \oplus_{\frac{1}{2}} x := 2)) \sqcap \mathsf{skip}, \\
R^\infty = [x = 1];\ \mathsf{abort} \sqcap [x = 0];\ \mathsf{magic}, \text{ and} \\
R^* \sqcap R^\infty = [x = 1];\ \mathsf{abort} \sqcap [x = 0];\ (x := 1 \oplus_{\frac{1}{2}} x := 2) \sqcap \mathsf{skip}.
\end{array}
$$

Since we equate program non-termination with abortion, the infinite iteration operator is not as interesting as the other two, and so in the remainder of this paper we focus on constructing transformation rules for the weak and strong iteration operators.

For a conjunctive predicate transformer R, we also have that the strong iteration operator may be expressed in terms of the weak iteration operator as follows [5]:

$$R^\omega = \{R^\omega.\mathsf{True}\};\ R^*.$$

Again, this relationship does not hold in general for sublinear expectation transformers. Using our previous example, we can see that

$$\begin{array}{ll} & \{R^\omega.\mathsf{True}\};\ R^* \\ = & \{\lambda\,\sigma \bullet (\sigma.x \notin \{0,1\}) \times 1 + (\sigma.x = 1) \times 0 + (\sigma.x = 0) \times \tfrac{1}{2}\};\ R^* \\ = & [x = 1];\ \mathsf{abort} \sqcap [x = 0];\ (\mathsf{abort}_{\frac{1}{2}} \oplus ((x := 1_{\frac{1}{2}} \oplus x := 2) \sqcap \mathsf{skip})) \sqcap [x \notin \{0,1\}]. \end{array}$$

3.2 Generalised Induction Properties

The following lemma and theorem may be used to specify more general induction rules.

Lemma 9. *Given monotonic expectation transformers S and T,*

$$S^\omega;\ T = (\mu\, X \bullet S;\ X \sqcap T) \text{ and}$$
$$S^*;\ T = (\nu X \bullet S;\ X \sqcap T).$$

Proof. The proof presented by Back and von Wright [5] applies to monotonic expectation transformers. □

For conjunctive expectation transformers S and T, we have that

$$T;\ S^* = (\nu X \bullet X;\ S \sqcap T)$$

holds, but this equivalence does not hold in general for sublinear expectation transformers, so we present an alternative theorem.

Theorem 10. *Let S and T be monotonic, cocontinuous expectation transformers. Then*

$$T;\ S^* = (\nu X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T).$$

Proof. First we prove

$$(\forall\, i : \mathbb{N} \bullet (\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)^{i+1}.\mathsf{magic} = T;\ (S \sqcap \mathsf{skip})^i)$$

by induction.

(a) Base case: $i = 0$

$$\begin{array}{ll} & (\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)^1.\mathsf{magic} \\ = & \{\text{function application}\} \\ & \mathsf{magic};\ (S \sqcap \mathsf{skip}) \sqcap T \\ = & \{\text{preemption and magic is unit}\} \\ & T \\ = & \{\mathsf{skip} \text{ is unit}\} \\ & T;\ \mathsf{skip} \\ = & \{\text{definition}\} \\ & T;\ (S \sqcap \mathsf{skip})^0 \end{array}$$

(b) Inductive case:
for any $i \in \mathbb{N}$ assume $(\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)^{i+1}.\mathsf{magic} = T;\ (S \sqcap \mathsf{skip})^i$
$(\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)^{i+2}.\mathsf{magic}$
$=$ {definition}
$(\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T).((\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)^{i+1}.\mathsf{magic})$
$=$ {inductive assumption}
$(\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T).(T;\ (S \sqcap \mathsf{skip})^i)$
$=$ {function application}
$T;\ (S \sqcap \mathsf{skip})^i;\ (S \sqcap \mathsf{skip}) \sqcap T$
$=$ {definition}
$T;\ (S \sqcap \mathsf{skip})^{i+1} \sqcap T$
$=$ {$T;\ (S \sqcap \mathsf{skip})^{i+1} \sqsubseteq T;\ \mathsf{skip}^{i+1} = T;\ \mathsf{skip} = T$, and general lattice rule
$x \sqsubseteq y \Rightarrow x \sqcap y = x$}
$T;\ (S \sqcap \mathsf{skip})^{i+1}$

We now prove our main result.

$(\nu X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)$
$=$ {Lemma 7 and cocontinuity of $(\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)$
follows from left distributivity of monotonic expectation transformers}
$(\sqcap i : \mathbb{N} \bullet (\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)^i.\mathsf{magic})$
$=$ {property of limits over descending chains}
$(\sqcap i : \mathbb{N} \bullet (\lambda\, X \bullet X;\ (S \sqcap \mathsf{skip}) \sqcap T)^{i+1}.\mathsf{magic})$
$=$ {see above}
$(\sqcap i : \mathbb{N} \bullet T;\ (S \sqcap \mathsf{skip})^i)$
$=$ {$\{i : \mathbb{N} \bullet (S \sqcap \mathsf{skip})^i\}$ is a $\sqsubseteq$-codirected set, cocontinuity of T, and Theorem 3}
$T;\ (\sqcap i : \mathbb{N} \bullet (S \sqcap \mathsf{skip})^i)$
$=$ {cocontinuity of S and Theorem 8}
$T;\ S^*$

□

There is no equivalent theorem to Theorem 10 for the strong iteration operator.

Theorem 11 (general induction). *Let R, S and T be monotonic expectation transformers, then*

$$S;\ X \sqcap R \sqsubseteq X \Rightarrow S^\omega;\ R \sqsubseteq X, \tag{1}$$

$$X \sqsubseteq T;\ X \sqcap R \Rightarrow X \sqsubseteq T^*;\ R, \text{ and} \tag{2}$$

$$X \sqsubseteq X;\ (T \sqcap \mathsf{skip}) \sqcap R \Rightarrow X \sqsubseteq R;\ T^*, \text{ if } T \text{ and } R \text{ cocontinuous.} \tag{3}$$

Proof. The first two properties are consequences of Lemma 9 and induction. The third follows from cocontinuity of T and R, Theorem 10 and induction. □

3.3 Basic Properties of Iterations

The following properties of iterations hold for both predicate and expectation transformers [5]:

Lemma 12. *For monotonic expectation transformer S,*

$$S \sqsubseteq T \Rightarrow S^\omega \sqsubseteq T^\omega \text{ and } S \sqsubseteq T \Rightarrow S^* \sqsubseteq T^*, \tag{4}$$
$$S^\omega \sqsubseteq S \text{ and } S^* \sqsubseteq S, \tag{5}$$
$$S^\omega \sqsubseteq \mathsf{skip} \text{ and } S^* \sqsubseteq \mathsf{skip}, \tag{6}$$
$$S^\omega;\ S^\omega = S^\omega \text{ and } S^*;\ S^* = S^*, \tag{7}$$
$$(S^\omega)^\omega = \mathsf{abort} \text{ and } (S^\omega)^* = S^\omega, \tag{8}$$
$$(S^*)^\omega = \mathsf{abort} \text{ and } (S^*)^* = S^*, \text{ and} \tag{9}$$
$$S^\infty = S^\omega;\ \mathsf{magic}. \tag{10}$$

Proof. The proofs provided by Back and von Wright [5] for these properties are valid here. They do not require any properties not satisfied by monotonic expectation transformers. □

The *decomposition* property also holds for monotonic expectation transformers (note that we do not require conjunctivity for this proof, we require left, but not right distributivity, which is implied by monotonicity alone (Fig. 4)).

Lemma 13 (decomposition). *For monotonic expectation transformers R and S,*

$$(R \sqcap S)^\omega = R^\omega;\ (S;\ R^\omega)^\omega \text{ and } (R \sqcap S)^* = R^*;\ (S;\ R^*)^*.$$

Proof. See Back and von Wright [5]. □

The *leapfrog* property [5] is valid for conjunctive expectation transformers, but not for all sublinear expectation transformers. For monotonic expectation transformers we have a weaker result.

Theorem 14 (leapfrog). *For monotonic expectation transformers R and S.*

$$R;\ (S;\ R)^\omega \sqsubseteq (R;\ S)^\omega;\ R \text{ and } R;\ (S;\ R)^* \sqsubseteq (R;\ S)^*;\ R.$$

In addition, if R is conjunctive then

$$R;\ (S;\ R)^\omega = (R;\ S)^\omega;\ R \text{ and } R;\ (S;\ R)^* = (R;\ S)^*;\ R.$$

Proof. The proof for strong iteration is as follows: it is very similar to the proof of Back and von Wright [5]. The proof for weak iteration is similar.

$R;\ (S;\ R)^\omega$
$=$ {definition}
$R;\ (\mu\, X \bullet S;\ R;\ X \sqcap \mathsf{skip})$
$=$ {rolling (Lemma 5) with $f \triangleq (\lambda\, X \bullet R;\ X)$ and $g \triangleq (\lambda\, X \bullet S;\ X \sqcap \mathsf{skip})$}
$(\mu\, X \bullet R;\ (S;\ X \sqcap \mathsf{skip}))$
$\sqsubseteq$ {right sub-distributivity and for any functions f and g,
$f \sqsubseteq g \Rightarrow (\mu\, X \bullet f.X) \sqsubseteq (\mu\, X \bullet g.X)$}
$(\mu\, X \bullet R;\ S;\ X \sqcap R)$
$=$ {Lemma 9}
$(R;\ S)^\omega;\ R$

If R is conjunctive the third step is an equality. □

Corollary 15. *For monotonic expectation transformer R, we have that*

$$R;\ R^\omega \sqsubseteq R^\omega;\ R \text{ and } R;\ R^* \sqsubseteq R^*;\ R.$$

Proof. This follows directly from Theorem 14 with $S = \mathsf{skip}$. □

Note that for sublinear (and hence monotonic) expectation transformer S, we do not necessarily have $S;\ S^\omega = S^\omega;\ S$, or that $S;\ S^* = S^*;\ S$. For example, take $S \triangleq [x = 0];\ (\mathsf{skip}_{\frac{1}{2}} \oplus x := 1)$, we then have that

$S^\omega = [x = 0];\ (\mathsf{skip} \sqcap x := 1) \sqcap [x \neq 0]$,
$S;\ S^\omega = [x = 0];\ ((x := 1 \sqcap \mathsf{skip})_{\frac{1}{2}} \oplus x := 1)$, and
$S^\omega;\ S = [x = 0];\ (\mathsf{skip}_{\frac{1}{2}} \oplus x := 1)$.

From a start state in which x is 0, S^ω may either skip or it may iterate until it assigns x to 1, or it may do some probabilistic combination of these behaviours: it is possible for S^ω to assign x to the value 1 because on each iteration of the loop it has a constant, non-zero probability of assigning x to 1.

3.4 Commutativity Properties

In this section we describe how commutativity properties are inherited by iterations. Such properties are useful when reasoning about data refinements of iterations.

Theorem 16. *Let R, S, and T be monotonic expectation transformers,*

$$R;\ S \sqsubseteq T;\ R \Rightarrow R;\ S^* \sqsubseteq T^*;\ R, \tag{11}$$
$$R;\ S \sqsubseteq T;\ R \Rightarrow R;\ S^\omega \sqsubseteq T^\omega;\ R, \text{ if } R \text{ is continuous,} \tag{12}$$
$$S;\ R \sqsubseteq R;\ T \Rightarrow S^*;\ R \sqsubseteq R;\ T^*, \text{ if } R \text{ is conjunctive,} \tag{13}$$
$$S;\ R \sqsubseteq R;\ T \Rightarrow S^\omega;\ R \sqsubseteq R;\ T^\omega, \text{ if } R \text{ is conjunctive,} \tag{14}$$
$$S;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip}) \Rightarrow S^*;\ R \sqsubseteq R;\ T^*, \text{ if } R \text{ and } T \text{ are cocontinuous, and} \tag{15}$$
$$S;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip}) \Rightarrow S^\omega;\ R \sqsubseteq R;\ T^\omega. \tag{16}$$

Proof. The proofs for the first four commutativity rules have been verified by Back and von Wright in the standard case [5]: the proofs for these do not require any properties that are not satisfied by monotonic expectation transformers. We focus on proving the last two rules because they differ from the usual rules for conjunctive predicate transformers. Assume R, S, and T are monotonic expectation transformers.

Proof of (15): Assume R and T are cocontinuous, then

$S^*;\ R \sqsubseteq R;\ T^*$
$\Leftarrow$ {cocontinuity of T and R, general induction (Theorem 11(3))}
$S^*;\ R \sqsubseteq S^*;\ R;\ (T \sqcap \mathsf{skip}) \sqcap R$

$\Leftrightarrow$ {unfolding and left distributivity}
  $S;\ S^*;\ R \sqcap R \sqsubseteq S^*;\ R;\ (T \sqcap \mathsf{skip}) \sqcap R$
$\Leftarrow$ {Corollary 15}
  $S^*;\ S;\ R \sqcap R \sqsubseteq S^*;\ R;\ (T \sqcap \mathsf{skip}) \sqcap R$
$\Leftarrow$ {monotonicity}
  $S;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip})$.

Proof of (16): Assume $S;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip})$. First we have that

  $S^\omega;\ R \sqsubseteq R;\ T^\omega$
$\Leftarrow$ {general induction (Theorem 11(1))}
  $S;\ R;\ T^\omega \sqcap R \sqsubseteq R;\ T^\omega$.

Refining the LHS,

  $S;\ R;\ T^\omega \sqcap R$
$\sqsubseteq$ {general lattice property $x \sqcap y \sqsubseteq x$}
  $S;\ R;\ T^\omega$
$\sqsubseteq$ {assumption}
  $R;\ (T \sqcap \mathsf{skip});\ T^\omega$
$\sqsubseteq$ {general lattice property $x \sqcap y \sqsubseteq x$, and skip is unit}
  $R;\ T^\omega$.

□

For monotonic expectation transformers, Theorem 16 parts (13) and (14) do not hold in general if R is sublinear$^+$ (and not conjunctive). For example, take

$S \triangleq [x = 0];\ (x, z := 1, 1 \,{}_{\frac{1}{2}}\!\oplus \mathsf{abort})$,
$T \triangleq [x = 0];\ x, z := 1, 2$, and
$R \triangleq [x = 0];\ (\mathsf{skip} \,{}_{\frac{1}{2}}\!\oplus x := 1) \sqcap [x \neq 0]$.

Note that S, R, and T, are sublinear. Then we have that

  $S;\ R \sqsubseteq R;\ T$
$\Leftrightarrow [x = 0];\ (x, z := 1, 1 \,{}_{\frac{1}{2}}\!\oplus \mathsf{abort}) \sqsubseteq [x = 0];\ (x, z := 1, 2 \,{}_{\frac{1}{2}}\!\oplus \mathsf{magic})$
$\Leftrightarrow \mathsf{true}$.

However,

$S^\omega = ([x = 0];\ (x, z := 1, 1 \,{}_{\frac{1}{2}}\!\oplus \mathsf{abort})) \sqcap \mathsf{skip}$,
$T^\omega = ([x = 0];\ x, z := 1, 2) \sqcap \mathsf{skip}$,
$S^\omega;\ R = [x = 0];\ ((x, z := 1, 1 \,{}_{\frac{1}{2}}\!\oplus \mathsf{abort}) \sqcap (\mathsf{skip} \,{}_{\frac{1}{2}}\!\oplus x := 1)) \sqcap [x \neq 0]$, and
$R;\ T^\omega = [x = 0];\ ((x, z := 1, 2 \sqcap \mathsf{skip}) \,{}_{\frac{1}{2}}\!\oplus x := 1) \sqcap [x \neq 0]$.

Hence we have that $S^\omega;\ R$ is not refined by $R;\ T^\omega$. To see that this is the case, take initial state σ in which variable x has value 0 and z has value 0, and expectation $\phi = (\lambda\, \sigma \bullet (\sigma.z = 0 \wedge \sigma.x = 0) \vee (\sigma.z = 1 \wedge \sigma.x = 1))$. We can see that the expectation of ϕ from σ in $S^\omega;\ R$ is $\frac{1}{2}$, however the expectation of ϕ from σ in $R;\ T^\omega$ is 0. We can also show that $S^*;\ R$ is not refined by $R;\ T^*$.

The following two rules, Theorems 17 and 18, are used later to verify common special cases of our data refinement rules for probabilistic action systems. They are more general than commutativity Theorem 16 parts (13) and (15), respectively. To see this observe that from Lemma 12 (5) we have that

$$R;\ S \sqsubseteq T;\ R \Rightarrow R;\ S^* \sqsubseteq T;\ R \text{ and}$$
$$S;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip}) \Rightarrow S^*;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip}).$$

Theorem 17. *Given monotonic expectation transformers R, S, and T,*

$$R;\ S^* \sqsubseteq T;\ R \Rightarrow R;\ S^* \sqsubseteq T^*;\ R.$$

Proof.
$R;\ S^* \sqsubseteq T;\ R$
⇒ {monotonicity}
$R;\ S^*;\ S^* \sqsubseteq T;\ R;\ S^*$
⇔ {Lemma 12 (7)}
$R;\ S^* \sqsubseteq T;\ R;\ S^*$
⇒ {basic lattice properties, and from Lemma 12 (6) we have that $R;\ S^* \sqsubseteq R$}
$R;\ S^* \sqsubseteq T;\ R;\ S^* \sqcap R$
⇒ {general induction (Theorem 11(2))}
$R;\ S^* \sqsubseteq T^*;\ R$
□

Theorem 18. *Given a monotonic expectation transformer S and monotonic and cocontinuous expectation transformers R and T,*

$$(S^*;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip})) \Rightarrow (S^*;\ R \sqsubseteq R;\ T^*).$$

Proof.
$S^*;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip})$
⇒ {monotonicity}
$S^*;\ S^*;\ R \sqsubseteq S^*;\ R;\ (T \sqcap \mathsf{skip})$
⇔ {Lemma 12 (7)}
$S^*;\ R \sqsubseteq S^*;\ R;\ (T \sqcap \mathsf{skip})$
⇒ {basic lattice properties, and from Lemma 12(6) we have that $S^*;\ R \sqsubseteq R$}
$S^*;\ R \sqsubseteq S^*;\ R;\ (T \sqcap \mathsf{skip}) \sqcap R$
⇒ {cocontinuity of R and T, general induction (Theorem 11(3))}
$S^*;\ R \sqsubseteq R;\ T^*$
□

Note that although $R;\ S^* \sqsubseteq T;\ R$ is equivalent to $R;\ S^* \sqsubseteq (T \sqcap \mathsf{skip});\ R$, condition $S^*;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip})$ is not equivalent to $S^*;\ R \sqsubseteq R;\ T$.

Next we present commutativity rules for guarded loops. Theorems 21, 22 and 23 are used in Sect. 4.2 to verify transformation rules for action systems, and in Sect. 5 to verify transformation rules for while-loops. It is possible to generate commutativity rules for guarded loops straight from the commutativity rule for iterations (Theorem 16). For example, take the following two theorems.[6]

[6] For standard guarded loops, Theorem 19 appears in the work of Back and von Wright [5, 24].

Theorem 19. *Given monotonic expectation transformers R, S and T such that R is continuous, and predicates g and p, we have that*

$$R;\ S^\omega;\ [g] \sqsubseteq T^\omega;\ [p];\ R,$$

provided

$$R;\ S \sqsubseteq T;\ R \text{ and} \tag{17}$$
$$R;\ [g] \sqsubseteq [p];\ R. \tag{18}$$

Proof.
$R;\ S^\omega;\ [g]$
$\sqsubseteq$ {assumption (17), Theorem 16 (12) and R is continuous}
$T^\omega;\ R;\ [g]$
$\sqsubseteq$ {assumption (18)}
$T^\omega;\ [p];\ R$
□

Theorem 20. *Given monotonic expectation transformers S, R and T, and predicates g and p, we have that*

$$S^\omega;\ [g];\ R \sqsubseteq R;\ T^\omega;\ [p],$$

provided

$$S;\ R \sqsubseteq R;\ (T \sqcap \mathsf{skip}) \text{ and} \tag{19}$$
$$[g];\ R \sqsubseteq R;\ [p]. \tag{20}$$

Proof.
$S^\omega;\ [g];\ R$
$\sqsubseteq$ {assumption (20)}
$S^\omega;\ R;\ [p]$
$\sqsubseteq$ {assumption (19), Theorem 16 (16)}
$R;\ T^\omega;\ [p]$
□

However, it is possible to generate a rule that is more general than Theorem 19 by taking into consideration the failure condition of the iteration body. Before we introduce such a rule we define some necessary terminology. Given an expectation transformer S, we refer to the set of states from which S may abort with probability one as $\mathsf{fail}.S$, where

$$\mathsf{fail}.S \triangleq (\lambda\, \sigma \bullet S.(\lambda\, \sigma \bullet \infty).\sigma = 0).$$

Our new rule, which does not appear in the work of Back and von Wright [5, 24], is then given as follows. This new theorem is more general than Theorem 19 since $[g \vee \mathsf{fail}.S] \sqsubseteq [g]$, hence

$$(R;\ [g] \sqsubseteq [p];\ R) \Rightarrow (R;\ [g \vee \mathsf{fail}.S] \sqsubseteq [p];\ R).$$

Theorem 21. *Given monotonic expectation transformers R, S and T such that R is continuous, and predicates g and p, we have that*

$$R;\ S^\omega;\ [g] \sqsubseteq T^\omega;\ [p];\ R,$$

provided

$$R;\ S \sqsubseteq T;\ R \text{ and} \tag{21}$$
$$R;\ [g \vee \mathsf{fail}.S] \sqsubseteq [p];\ R. \tag{22}$$

Proof.

$R;\ S^\omega;\ [g] \sqsubseteq T^\omega;\ [p];\ R$
$\Leftrightarrow$ {Lemma 9}
$R;\ (\mu\, X \bullet S;\ X \sqcap [g]) \sqsubseteq (\mu\, X \bullet T;\ X \sqcap [p];\ R)$
$\Leftarrow$ {from continuity of R, $(\lambda\, X \bullet R;\ X)$ is continuous, fusion (Lemma 6)}
$(\lambda\, X \bullet R;\ X) \circ (\lambda\, X \bullet S;\ X \sqcap [g]) \sqsubseteq (\lambda\, X \bullet T;\ X \sqcap [p];\ R) \circ (\lambda\, X \bullet R;\ X)$
$\Leftrightarrow$ {function application}
$(\lambda\, X \bullet R;\ (S;\ X \sqcap [g])) \sqsubseteq (\lambda\, X \bullet T;\ R;\ X \sqcap [p];\ R)$

And we have that for any expectation transformer X, in order to show

$$R;\ (S;\ X \sqcap [g]) \sqsubseteq T;\ R;\ X \sqcap [p];\ R,$$

it is sufficient to show the following.

(a) $R;\ (S;\ X \sqcap [g]) \sqsubseteq T;\ R;\ X$

$R;\ (S;\ X \sqcap [g])$
$\sqsubseteq$ {general lattice property $x \sqcap y \sqsubseteq x$}
$R;\ S;\ X$
$\sqsubseteq$ {assumption (21)}
$T;\ R;\ X$

(b) $R;\ (S;\ X \sqcap [g]) \sqsubseteq [p];\ R$

First we prove

$S;\ X$
$\sqsubseteq$ {basic guard rule $\mathsf{skip} \sqsubseteq [g]$}
$[\mathsf{fail}.S];\ S;\ X$
$=$ {basic guard and assumption rules}
$[\mathsf{fail}.S];\ \{\mathsf{fail}.S\};\ S;\ X$
$=$ {from the definition of fail, $\{\mathsf{fail}.S\};\ S = \mathsf{abort}$}
$[\mathsf{fail}.S];\ \mathsf{abort};\ X$
$\sqsubseteq$ {preemption, abort is least element}
$[\mathsf{fail}.S]$.

We then have that,

$R;\ (S;\ X \sqcap [g])$
$\sqsubseteq$ {$S;\ X \sqsubseteq [\mathsf{fail}.S]$, see above}
$R;\ ([\mathsf{fail}.S] \sqcap [g])$
$=$ {basic guard rules}
$R;\ [g \vee \mathsf{fail}.S]$
$\sqsubseteq$ {assumption (22)}
$[p];\ R$.

□

The proof obligations in Theorem 20 seem to be quite restrictive, for example, condition (19) requires $S;\ R$ to be refined by R. For guarded loops of the specific form $([g];\ S)^\omega;\ [\neg g]$, it is possible to construct an alternative rule. As suggested by the commutativity rule for iterations, Theorem 16 (16), the conditions required to prove a refinement of the form $([g];\ S)^\omega;\ [\neg g];\ R \sqsubseteq R;\ ([p];\ T)^\omega;\ [\neg p]$, for monotonic expectation transformers S, R, and T, differ from those that one would normally expect for the case when R is conjunctive.

Theorem 22. *Given monotonic expectation transformer R, S and T, and predicates g and p, we have that*

$$([g];\ S)^\omega;\ [\neg g];\ R \sqsubseteq R;\ ([p];\ T)^\omega;\ [\neg p],$$

provided

$$\{g\};\ S;\ R \sqsubseteq R;\ \{p\};\ T \text{ and} \tag{23}$$
$$R;\ [\neg p] \sqsubseteq [\neg g];\ R. \tag{24}$$

Proof. This rule is a special case of the following theorem, Theorem 23. □

For proving refinements of the form $([g];\ S)^\omega;\ [\neg g];\ R \sqsubseteq R;\ ([p];\ T)^\omega;\ [\neg p]$, this rule appears to have a more useful form than Theorem 20. (Although we cannot say that Theorem 20 is weaker than Theorem 22 because the preconditions in Theorem 20 do not imply those in Theorem 22.) The following theorem is slightly more general than Theorem 22, it is applicable to guarded loops of the form $([p1];\ S)^\omega;\ [p2]$, where $p1 \Rightarrow \neg p2$. It is used later to give an alternative stuttering sensitive action system data refinement rule (Theorem 34).

Theorem 23. *Given monotonic expectation transformer R, S and T, and predicates g and p, we have that*

$$([g1];\ S)^\omega;\ [g2];\ R \sqsubseteq R;\ ([p1];\ T)^\omega;\ [p2],$$

provided

$$\{\neg g2\};\ S;\ R \sqsubseteq R;\ \{\neg p2\};\ T, \tag{25}$$
$$R;\ [\neg p1] \sqsubseteq [g2];\ R, \tag{26}$$
$$[g1 \vee g2];\ R \sqsubseteq R;\ [p1 \vee p2], \tag{27}$$
$$g1 \Rightarrow \neg g2, \text{ and} \tag{28}$$
$$p1 \Rightarrow \neg p2. \tag{29}$$

Proof.

$([g1];\ S)^\omega;\ [g2];\ R \sqsubseteq R;\ ([p1];\ T)^\omega;\ [p2]$
$\Leftarrow$ {general induction (Theorem 11 (1))}
$[g1];\ S;\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R \sqsubseteq R;\ ([p1];\ T)^\omega;\ [p2]$

Refining the LHS,

$[g1];\ S;\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R$
$=$ {basic guard and assertion rules and assumption (28)}
$[g1];\ \{\neg g2\};\ S;\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R$
$\sqsubseteq$ {assumption (25)}
$[g1];\ R;\ \{\neg p2\};\ T;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R$
$=$ {basic guard rule $\{p\} = [\neg p];\ \mathsf{abort} \sqcap [p]$, left distributivity and preemption}
$[g1];\ R;\ ([\neg p2];\ T;\ ([p1];\ T)^\omega;\ [p2] \sqcap [p2];\ \mathsf{abort}) \sqcap [g2];\ R$
$\sqsubseteq$ {assumption (29), basic guard rules, abort is bottom}
$[g1];\ R;\ ([p1];\ T;\ ([p1];\ T)^\omega;\ [p2] \sqcap [p2]) \sqcap [g2];\ R$
$=$ {left distributivity and unfolding}
$[g1];\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R$
$\sqsubseteq$ {basic guard rule $\mathsf{skip} \sqsubseteq [g]$}
$[g1];\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R;\ [\neg p1]$
$=$ {from the basic guard rules $[\neg p];\ [p] = \mathsf{magic}$ and magic is unit}
$[g1];\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R;\ ([\neg p1];\ [p1];\ R;\ ([p1];\ T)^\omega \sqcap [\neg p1])$
$=$ {right and left distributivity of conjunctive programs}
$[g1];\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R;\ [\neg p1];\ ([p1];\ R;\ ([p1];\ T)^\omega \sqcap \mathsf{skip});\ [\neg p1]$
$=$ {unfolding}
$[g1];\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R;\ [\neg p1];\ ([p1];\ T)^\omega;\ [\neg p1]$
$\sqsubseteq$ {assumptions (26) and (29), basic guard rules}
$[g1];\ R;\ ([p1];\ T)^\omega;\ [p2] \sqcap [g2];\ R;\ ([p1];\ T)^\omega;\ [p2]$
$=$ {left distributivity and basic guard rules}
$[g1 \vee g2];\ R;\ ([p1];\ T)^\omega;\ [p2]$
$\sqsubseteq$ {assumption (27)}
$R;\ [p1 \vee p2];\ ([p1];\ T)^\omega;\ [p2]$
$=$ {unfolding}
$R;\ [p1 \vee p2];\ ([p1];\ T;\ ([p1];\ T)^\omega \sqcap \mathsf{skip});\ [p2]$
$=$ {right distributivity of guards, left distributivity}
$R;\ ([p1 \vee p2];\ [p1];\ T;\ ([p1];\ T)^\omega;\ [p2] \sqcap [p1 \vee p2];\ [p2])$
$=$ {basic guard rules}
$R;\ ([p1];\ T;\ ([p1];\ T)^\omega;\ [p2] \sqcap [p2])$
$=$ {left distributivity and unfolding}
$R;\ ([p1];\ T)^\omega;\ [p2].$

□

The other kind of commutativity law presented by Back and von Wright [5] refers to a special kind of commutation: S is said to *commute over* T if $S;\ T \sqsubseteq T;\ S$. The following lemma (originally for predicate transformers [5]) also applies to expectation transformers given a *conjunctivity* assumption on T, however, it is important to note that it does not hold for the general case that T is sublinear instead of conjunctive. Note that for any monotonic S and T, $(S \sqcap T)^\omega \sqsubseteq S^\omega;\ T^\omega$.

Lemma 24. *Assume that S is monotonic, T is conjunctive, and $S;\ T \sqsubseteq T;\ S$. Then*

$$S^\omega;\ T \sqsubseteq T;\ S^\omega, \tag{30}$$

$$(S \sqcap T)^\omega = S^\omega;\ T^\omega, \quad \textit{if } S \textit{ is continuous, and} \tag{31}$$

$$(S \sqcap T)^\omega = S^\omega;\ T^\omega,\ \ \mathit{if}\ T^\omega = T^*. \tag{32}$$

Proof. Refer to Back and von Wright [5]. □

We can demonstrate that this lemma does not hold when T is not conjunctive. For (30) chose S and T such that S equals T, we then have that $S;\ S \sqsubseteq S;\ S$, but from the Corollary 15 we have that $S;\ S^\omega \sqsubseteq S^\omega;\ S$, but not necessarily $S^\omega;\ S \sqsubseteq S;\ S^\omega$. In Sect. 3.3 we presented a counter example that demonstrated this. For (31) and (32) let,

$S \triangleq [(y = 0 \lor y = 1) \land x = 0];\ x := 1$ and
$T \triangleq [y = 0];\ (y := 1_{\frac{1}{2}} \oplus y = 2)$.

We then have that S is continuous and $T^\omega = T^*$ and

$$\begin{array}{l} \quad S;\ T \\ = [y = 0 \land x = 0];\ (x, y := 1, 1_{\frac{1}{2}} \oplus x, y := 1, 2) \\ \sqsubseteq [y = 0 \land x = 0];\ (x, y := 1, 1_{\frac{1}{2}} \oplus \mathsf{magic}) \\ = T;\ S, \end{array}$$

however,

$$\begin{array}{lll} S^\omega & = [(y = 0 \lor y = 1) \land x = 0];\ x := 1 \sqcap \mathsf{skip}, \\ T^\omega & = [y = 0];\ (y := 1_{\frac{1}{2}} \oplus y := 2) \sqcap \mathsf{skip}, \\ S^\omega;\ T^\omega & = [y = 0 \land x = 0];\ ((x, y := 1, 1_{\frac{1}{2}} \oplus x, y := 1, 2) \sqcap x := 1 \\ & \qquad\qquad \sqcap (y := 1_{\frac{1}{2}} \oplus y := 2) \sqcap \mathsf{skip}) \\ & \quad \sqcap [y = 1 \land x = 0];\ (x := 1 \sqcap \mathsf{skip}) \\ & \quad \sqcap [y = 0 \land x \neq 0];\ ((y := 1_{\frac{1}{2}} \oplus y := 2) \sqcap \mathsf{skip}) \\ & \quad \mathsf{skip}, \text{ and} \\ (S \sqcap T)^\omega & = [y = 0 \land x = 0];\ ((x, y := 1, 1_{\frac{1}{2}} \oplus x, y := 1, 2) \sqcap x := 1 \\ & \qquad\qquad \sqcap ((x, y := 1, 1 \sqcap y := 1)_{\frac{1}{2}} \oplus y := 2) \sqcap \mathsf{skip}) \\ & \quad \sqcap [y = 1 \land x = 0];\ (x := 1 \sqcap \mathsf{skip}) \\ & \quad \sqcap [y = 0 \land x \neq 0];\ ((y := 1_{\frac{1}{2}} \oplus y := 2) \sqcap \mathsf{skip}) \\ & \quad \sqcap \mathsf{skip}. \end{array}$$

Hence $(S \sqcap T)^\omega$ is not a refinement of $S^\omega;\ T^\omega$.

4 Action Systems

So far we have investigated properties of the general iteration constructs. We now follow the lead of Back and von Wright [5] by applying these results to more well known and useful programming constructs: namely action systems. Action systems can be used to model parallel or distributed systems in which concurrent behaviour is modeled by interleaving atomic actions [3, 4]. Probabilistic action systems (originally proposed by Sere and Troubitsyna [21, 23]) are an extension of standard action systems in which actions are defined to be

sublinear$^+$ expectation transformers instead of conjunctive predicate transformers. The input/output behaviour of a probabilistic action system is defined in terms of the iteration constructs as follows:

$$\mathsf{do}\ A_1 \sqcap ... \sqcap A_n\ \mathsf{od} \triangleq (A_1 \sqcap ... \sqcap A_n)^\omega;\ [\neg \mathsf{gd}.A_1 \wedge ... \wedge \neg \mathsf{gd}.A_n],$$

where for all $1 \leq i \leq n$, the *action* A_i is a sublinear$^+$ expectation transformer, and $\mathsf{gd}.A$ is a predicate that specifies the set of states from which A does not behave like magic.

$$\mathsf{gd}.A \triangleq (\lambda\, \sigma \bullet A.\mathsf{False}.\sigma = 0)$$

The following lemma describes some of the basic properties of the guard operator.

Lemma 25. *For any sublinear$^+$ expectation transformers A and A_i, for $1 \leq i \leq n$, we have that,*

$$[\mathsf{gd}.A];\ A = A, \tag{33}$$

$$(\{\mathsf{gd}.A\};\ A).\mathsf{False} = \mathsf{False}, \textit{ and} \tag{34}$$

$$\mathsf{gd}.(A_1 \sqcap ... \sqcap A_n) = \mathsf{gd}.A_1 \vee .... \vee \mathsf{gd}.A_n. \tag{35}$$

Proof. Proof of (33):

$[\mathsf{gd}.A];\ A = A$
$\Leftrightarrow$ {refinement ordering}
$(\forall\, \phi \bullet ([\mathsf{gd}.A];\ A).\phi = A.\phi)$
$\Leftrightarrow$ {definition of sequential composition and guard}
$(\forall\, \phi \bullet (\lambda\, \sigma \bullet \mathsf{if}\ \mathsf{gd}.A.\sigma\ \mathsf{then}\ A.\phi.\sigma\ \mathsf{else}\ \infty) = A.\phi)$
$\Leftrightarrow$ {expectation ordering defined pointwise}
$(\forall\, \phi \bullet (\forall\, \sigma \bullet \mathsf{if}\ \mathsf{gd}.A.\sigma\ \mathsf{then}\ A.\phi.\sigma = A.\phi.\sigma\ \mathsf{else}\ \infty = A.\phi.\sigma))$
$\Leftrightarrow$ {simplify}
$(\forall\, \phi, \sigma \bullet \neg \mathsf{gd}.A.\sigma \Rightarrow (A.\phi.\sigma = \infty))$
$\Leftrightarrow$ {definition of gd}
$(\forall\, \phi, \sigma \bullet (A.\mathsf{False}.\sigma \neq 0) \Rightarrow (A.\phi.\sigma = \infty))$
$\Leftarrow$ {sublinear$^+$ expectation transformers satisfy 01-feasibility (Lemma 1)}
$(\forall\, \phi, \sigma \bullet (A.\mathsf{False}.\sigma = \infty) \Rightarrow (A.\phi.\sigma = \infty))$
$\Leftrightarrow$ { False is bottom expectation, monotonicity}
true.

Proof of (34):

$(\{\mathsf{gd}.A\};\ A).\mathsf{False}$
$=$ {definition of sequential composition}
$\{\mathsf{gd}.A\}.(A.\mathsf{False})$
$=$ {definition of assertion on predicate}
$(\lambda\, \sigma \bullet \mathsf{if}\ \mathsf{gd}.A.\sigma\ \mathsf{then}\ A.\mathsf{False}.\sigma\ \mathsf{else}\ 0)$
$=$ {definition of gd}
$(\lambda\, \sigma \bullet \mathsf{if}\ (A.\mathsf{False}.\sigma = 0)\ \mathsf{then}\ A.\mathsf{False}.\sigma\ \mathsf{else}\ 0)$
$=$ {simplify}
False.

Proof of (35):

$$
\begin{array}{ll}
& \mathsf{gd}.(A_1 \sqcap ... \sqcap A_n) \\
= & \{\text{definition of } \mathsf{gd}\} \\
& (\lambda\, \sigma \bullet (A_1 \sqcap ... \sqcap A_n).\mathsf{False}.\sigma = 0) \\
= & \{\text{definition of demonic choice, expectations are bounded below by } 0\} \\
& (\lambda\, \sigma \bullet (A_1.\mathsf{False}.\sigma = 0) \vee ... \vee (A_n.\mathsf{False}.\sigma = 0)) \\
= & \{\text{definition of } \mathsf{gd}\} \\
= & \mathsf{gd}.A_1 \vee .... \vee \mathsf{gd}.A_n.
\end{array}
$$

□

In this model of action systems infinite behaviours are considered to be aborting: we do not model reactive behaviour. Using our algebraic framework, we construct and verify transformation rules for probabilistic action systems. First we describe some basic properties, and then we consider data refinement of action systems in detail.

4.1 Basic Properties

Lemma 26. *For any sublinear$^+$ expectation transformer A,*

$$(\mathsf{do}\ A\ \mathsf{od}).\mathsf{False} = \mathsf{False}.$$

Proof. Since False is the least expectation it is sufficient to show that

$$
\begin{array}{ll}
& (\mathsf{do}\ A\ \mathsf{od}).\mathsf{False} \leq \mathsf{False} \\
\Leftrightarrow & \{\text{for all } \phi,\ \mathsf{abort}.\phi = \mathsf{False}\} \\
& \mathsf{do}\ A\ \mathsf{od};\ \mathsf{abort} \sqsubseteq \mathsf{abort} \\
\Leftrightarrow & \{\text{action system definition}\} \\
& A^{\omega};\ [\neg \mathsf{gd}.A];\ \mathsf{abort} \sqsubseteq \mathsf{abort} \\
\Leftarrow & (\text{general induction (Theorem 11(1))}) \\
& A;\ \mathsf{abort} \sqcap [\neg \mathsf{gd}.A];\ \mathsf{abort} \sqsubseteq \mathsf{abort} \\
\Leftrightarrow & \{\text{ Lemma 25 (33) and basic guard and assertion rules}\} \\
& [\mathsf{gd}.A];\ \{\mathsf{gd}.A\};\ A;\ \mathsf{abort} \sqcap [\neg \mathsf{gd}.A];\ \mathsf{abort} \sqsubseteq \mathsf{abort} \\
\Leftrightarrow & \{\{\mathsf{gd}.A\};\ A \text{ is strict (Lemma 25 (34))}\} \\
& [\mathsf{gd}.A];\ \mathsf{abort} \sqcap [\neg \mathsf{gd}.A];\ \mathsf{abort} \sqsubseteq \mathsf{abort} \\
\Leftrightarrow & \{\text{left distributivity and basic guard rules}\} \\
& [\mathsf{gd}.A \vee \neg \mathsf{gd}.A];\ \mathsf{abort} \sqsubseteq \mathsf{abort} \\
\Leftrightarrow & \{\text{definition of skip and skip is unit}\} \\
& \mathsf{true}.
\end{array}
$$

□

As for standard action systems, the leapfrog and decomposition properties may be lifted to action systems.

Lemma 27 (action system leapfrog). *Assume that R and S are sublinear$^+$ expectation transformers. If $R;\ [\neg \mathsf{gd}.(S;\ R)] \sqsubseteq [\neg \mathsf{gd}.(R;\ S)];\ R$, then*

$$R;\ \mathsf{do}\ S;\ R\ \mathsf{od} \sqsubseteq \mathsf{do}\ R;\ S\ \mathsf{od};\ R.$$

Proof. The proof of this is similar to that of Back and von Wright for standard programs [5], it differs because we have refinement in the second proof step instead of equivalence.

$$
\begin{array}{ll}
 & R;\ \mathsf{do}\ S;\ R\ \mathsf{od} \\
= & \{\text{rewrite action system in terms of iteration constructs}\} \\
 & R;\ (S;\ R)^{\omega};\ [\neg \mathsf{gd}.(S;\ R)] \\
\sqsubseteq & \{\text{sub-leapfrog (Theorem 14)}\} \\
 & (R;\ S)^{\omega};\ R;\ [\neg \mathsf{gd}.(S;\ R)] \\
\sqsubseteq & \{\text{assumption}\} \\
 & (R;\ S)^{\omega};\ [\neg \mathsf{gd}.(R;\ S)];\ R \\
= & \{\text{rewrite iteration construct in terms of action system}\} \\
 & \mathsf{do}\ R;\ S\ \mathsf{od};\ R
\end{array}
$$

□

Note that unlike standard action systems [5], property

$$R;\ [\neg \mathsf{gd}.(S;\ R)] \sqsubseteq [\neg \mathsf{gd}.(R;\ S)];\ R \tag{36}$$

does not necessarily hold. Take for example,

$R = (x := 0_{\frac{1}{2}} \oplus x := 1)$ and
$S = [x = 0]$.

We then have that

$S;\ R = [x = 0];\ (x := 0_{\frac{1}{2}} \oplus x := 1)$,
$R;\ S = (x := 0_{\frac{1}{2}} \oplus x := 1);\ [x = 0] = (x := 0_{\frac{1}{2}} \oplus \mathsf{magic}) = \mathsf{magic}$,
$R;\ [\neg \mathsf{gd}.(S;\ R)] = R;\ [x \neq 0] = (\mathsf{magic}_{\frac{1}{2}} \oplus x := 1) = \mathsf{magic}$, but
$[\neg \mathsf{gd}.(R;\ S)];\ R = [\mathsf{True}];\ R = R$.

The top program magic is not refined by R unless R is magic. Hence we must include (36) as an assumption in the action system leapfrog rule.

Lemma 28 (action system decomposition). *Let R and S be sublinear$^+$ expectation transformers such that* $\mathsf{gd}.R \wedge \mathsf{gd}.S = \mathsf{False}$*, then*

$$\mathsf{do}\ R \parallel S\ \mathsf{od} = \mathsf{do}\ S\ \mathsf{od};\ \mathsf{do}\ (R;\ \mathsf{do}\ S\ \mathsf{od})\ \mathsf{od}.$$

Proof. The proof of this lemma is similar to the proof by Back and von Wright [5] for the standard case. We include it here for clarity.
First, if we assume that $\mathsf{gd}.S \wedge \mathsf{gd}.R = \mathsf{False}$, then we have

$$
\begin{array}{ll}
 & R \\
= & \{\text{definition}\} \\
 & (\lambda\, \phi \bullet (\lambda\, \sigma \bullet R.\phi.\sigma)) \\
= & \{\mathsf{gd}.S \text{ is standard}\} \\
 & (\lambda\, \phi \bullet (\lambda\, \sigma \bullet \mathsf{if}\ \neg \mathsf{gd}.S.\sigma\ \mathsf{then}\ R.\phi.\sigma\ \mathsf{else}\ \mathsf{gd}.S.\sigma \wedge R.\phi.\sigma)) \\
= & \{\text{Lemma 25 (33)}\} \\
 & (\lambda\, \phi \bullet (\lambda\, \sigma \bullet \mathsf{if}\ \neg \mathsf{gd}.S.\sigma\ \mathsf{then}\ R.\phi.\sigma\ \mathsf{else}\ \mathsf{if}\ \mathsf{gd}.S.\sigma \wedge \mathsf{gd}.R.\sigma\ \mathsf{then}\ R.\phi.\sigma\ \mathsf{else}\ \infty)) \\
= & \{\text{assumption}\} \\
 & (\lambda\, \phi \bullet (\lambda\, \sigma \bullet \mathsf{if}\ \neg \mathsf{gd}.S.\sigma\ \mathsf{then}\ R.\phi.\sigma\ \mathsf{else}\ \mathsf{if}\ \mathsf{false}\ \mathsf{then}\ R.\phi.\sigma\ \mathsf{else}\ \infty)) \\
= & \{\text{simplify}\} \\
 & (\lambda\, \phi \bullet (\lambda\, \sigma \bullet \mathsf{if}\ \neg \mathsf{gd}.S.\sigma\ \mathsf{then}\ R.\phi.\sigma\ \mathsf{else}\ \infty)) \\
= & \{\text{definition}\} \\
 & [\neg \mathsf{gd}.S];\ R.
\end{array}
$$

Now,

$$
\begin{array}{ll}
 & \mathsf{do}\ R \parallel S\ \mathsf{od} \\
= & \{\text{rewrite action system using iteration constructs}\} \\
 & (R \sqcap S)^{\omega};\ [\neg \mathsf{gd}.R \wedge \neg \mathsf{gd}.S] \\
= & \{\text{decomposition (Lemma 13)}\} \\
 & S^{\omega};\ (R;\ S^{\omega})^{\omega};\ [\neg \mathsf{gd}.S \wedge \neg \mathsf{gd}.R] \\
= & \{\text{basic guard rules}\} \\
 & S^{\omega};\ (R;\ S^{\omega})^{\omega};\ [\neg \mathsf{gd}.S];\ [\neg \mathsf{gd}.R] \\
= & \{\text{above}\} \\
 & S^{\omega};\ ([\neg \mathsf{gd}.S];\ R;\ S^{\omega})^{\omega};\ [\neg \mathsf{gd}.S];\ [\neg \mathsf{gd}.R] \\
= & \{\text{guards are conjunctive, leapfrog (Theorem 14)}\} \\
 & S^{\omega};\ [\neg \mathsf{gd}.S];\ (R;\ S^{\omega};\ [\neg \mathsf{gd}.S])^{\omega};\ [\neg \mathsf{gd}.R] \\
= & \{\text{see separate derivation below}\} \\
 & S^{\omega};\ [\neg \mathsf{gd}.S];\ (R;\ S^{\omega};\ [\neg \mathsf{gd}.S])^{\omega};\ [\neg \mathsf{gd}.(R;\ S^{\omega};\ [\neg \mathsf{gd}.S])] \\
= & \{\text{rewrite iteration constructs as action systems}\} \\
 & \mathsf{do}\ S\ \mathsf{od};\ \mathsf{do}\ R;\ \mathsf{do}\ S\ \mathsf{od}\ \mathsf{od}.
\end{array}
$$

The guard manipulation step is justified by

$$
\begin{array}{ll}
 & \mathsf{gd}.(R;\ S^{\omega};\ [\neg \mathsf{gd}.S]) \\
= & \{\text{rewrite iteration in terms of action system}\} \\
 & \mathsf{gd}.(R;\ \mathsf{do}\ S\ \mathsf{od}) \\
= & \{\text{definition of } \mathsf{gd}\} \\
 & (\lambda\, \sigma \bullet (R;\ \mathsf{do}\ S\ \mathsf{od}).\mathsf{False}.\sigma = 0) \\
= & \{\text{definition of sequential composition}\} \\
 & (\lambda\, \sigma \bullet R.(\mathsf{do}\ S\ \mathsf{od}.\mathsf{False}).\sigma = 0) \\
= & \{\text{Lemma 26}\} \\
 & (\lambda\, \sigma \bullet R.\mathsf{False}.\sigma = 0) \\
= & \{\text{definition}\} \\
 & \mathsf{gd}.R.
\end{array}
$$

□

4.2 Data Refinement

An expectation transformer S is said to be data refined by T through R if either

$$R;\ S \sqsubseteq T;\ R \text{ or } S;\ R \sqsubseteq R;\ T.$$

In the first instance R can be seen as mapping from the concrete state of T to the abstract state of S, and in the second R can be seen to map the abstract state of S to the concrete state of T. We refer to data refinement in the first instance as *cosimulation*, and *simulation* in the latter.

First we present basic cosimulation and simulation rules for probabilistic action systems. These rules are *stuttering insensitive*, that is they require a direct correspondence between actions. The cosimulation rule has a similar form to the cosimulation data refinement rule for standard action systems [1]. The simulation rule has (necessarily) a different form to the corresponding standard action system rule. We demonstrate our simulation rule using a simple example. After we have presented the basic rules we present and verify *stuttering sensitive* versions of the rules: rules that do not require a direct correspondence between actions. After this we present and verify some weaker versions of our general rules: these weaker rules may be simpler to use in practice than our more general rules.

The basic cosimulation and simulation rules are as follows.

Theorem 29 (basic cosimulation). *Given sublinear$^+$ expectation transformers R, S and T, we have that $R;\ \mathsf{do}\ S\ \mathsf{od} \sqsubseteq \mathsf{do}\ T\ \mathsf{od};\ R$, if R is continuous,*

$$R;\ S \sqsubseteq T;\ R, \text{ and} \tag{37}$$

$$R;\ [\neg \mathsf{gd}.S \vee \mathsf{fail}.S] \sqsubseteq [\neg \mathsf{gd}.T];\ R. \tag{38}$$

Proof. This follows directly from the definition of action systems, assumptions (37) and (38), continuity of R, and Theorem 21. □

Theorem 30 (basic simulation). *Given sublinear$^+$ expectation transformers R, S and T, we have that $\mathsf{do}\ S\ \mathsf{od};\ R \sqsubseteq R;\ \mathsf{do}\ T\ \mathsf{od}$, if*

$$\{\mathsf{gd}.S\};\ S;\ R \sqsubseteq R;\ \{\mathsf{gd}.T\};\ T \text{ and} \tag{39}$$

$$R;\ [\neg \mathsf{gd}.T] \sqsubseteq [\neg \mathsf{gd}.S];\ R. \tag{40}$$

Proof. This follows directly from the definition of action systems, assumptions (39) and (40), and Theorem 22. □

Our basic cosimulation rule is more general than the one presented by Back and von Wright [5, 24] because it uses our more general commutativity rule, Theorem 21.

General Rules. We now present data refinement rules that allow *stuttering* actions in data refinement. We use the following lemma to generalise our previous rules.

Lemma 31. *Given action system $\mathsf{do}\ S_\sharp \sqcap S_\natural\ \mathsf{od}$, such that $S_\natural$ and $S_\sharp$ are sublinear$^+$ expectation transformers and ${S_\natural}^\omega = {S_\natural}^*$,*

$$\mathsf{do}\ S_\sharp \sqcap S_\natural\ \mathsf{od} = {S_\natural}^*;\ (S_\sharp;\ {S_\natural}^*)^\omega;\ [\neg \mathsf{gd}.S_\natural \sqcap \neg \mathsf{gd}.S_\sharp].$$

Proof.

$$
\begin{array}{ll}
 & \mathsf{do}\ S_\sharp \sqcap S_\natural\ \mathsf{od} \\
= & \{\text{rewrite action system in terms of iteration}\} \\
 & (S_\sharp \sqcap S_\natural)^\omega;\ [\neg \mathsf{gd}.S_\natural \sqcap \neg \mathsf{gd}.S_\sharp] \\
= & \{\text{decomposition (Lemma 13)}\} \\
 & {S_\natural}^\omega;\ (S_\sharp;\ {S_\natural}^\omega)^\omega;\ [\neg \mathsf{gd}.S_\natural \sqcap \neg \mathsf{gd}.S_\sharp] \\
= & \{\text{assumption}\} \\
 & {S_\natural}^*;\ (S_\sharp;\ {S_\natural}^*)^\omega;\ [\neg \mathsf{gd}.S_\natural \sqcap \neg \mathsf{gd}.S_\sharp]
\end{array}
$$

□

Our general cosimulation and simulation rules are then given as follows.

Theorem 32 (general cosimulation). *Given sublinear$^+$ expectation transformers R, $S = S_\sharp \sqcap S_\natural$ and $T = T_\sharp \sqcap T_\natural$ such that ${S_\natural}^\omega = {S_\natural}^*$ and ${T_\natural}^\omega = {T_\natural}^*$, we have that $R;\ \mathsf{do}\ S\ \mathsf{od} \sqsubseteq \mathsf{do}\ T\ \mathsf{od};\ R$, if R is continuous,*

$$R;\ {S_\natural}^* \sqsubseteq {T_\natural}^*;\ R, \tag{41}$$

$$R;\ S_\sharp;\ {S_\natural}^* \sqsubseteq T_\sharp;\ {T_\natural}^*;\ R, \text{ and} \tag{42}$$

$$R;\ [\neg \mathsf{gd}.S \vee \mathit{fail}.(S_\sharp;\ {S_\natural}^*)] \sqsubseteq [\neg \mathsf{gd}.T];\ R. \tag{43}$$

Proof.

$$
\begin{array}{ll}
 & R;\ \mathsf{do}\ S\ \mathsf{od} \\
= & \{\text{Lemma 31}\} \\
 & R;\ {S_\natural}^*;\ (S_\sharp;\ {S_\natural}^*)^\omega;\ [\neg \mathsf{gd}.S] \\
\sqsubseteq & \{\text{assumption (41)}\} \\
 & {T_\natural}^*;\ R;\ (S_\sharp;\ {S_\natural}^*)^\omega;\ [\neg \mathsf{gd}.S] \\
\sqsubseteq & \{\text{assumptions (42) and (43), continuity of } R\text{, and Theorem 21}\} \\
 & {T_\natural}^*;\ (T_\sharp;\ {T_\natural}^*)^\omega;\ [\neg \mathsf{gd}.T];\ R \\
= & \{\text{Lemma 31}\} \\
 & \mathsf{do}\ T\ \mathsf{od};\ R
\end{array}
$$

□

Theorem 33 (general simulation). *Given sublinear$^+$ expectation transformers R, $S = S_\sharp \sqcap S_\natural$ and $T = T_\sharp \sqcap T_\natural$ such that ${S_\natural}^\omega = {S_\natural}^*$ and ${T_\natural}^\omega = {T_\natural}^*$, we have that $\mathsf{do}\ S\ \mathsf{od};\ R \sqsubseteq R;\ \mathsf{do}\ T\ \mathsf{od}$, if*

$${S_\natural}^*;\ R \sqsubseteq R;\ {T_\natural}^*, \tag{44}$$

$$\{\mathsf{gd}.S\};\ S_\sharp;\ {S_\natural}^*;\ R \sqsubseteq R;\ \{\mathsf{gd}.T\};\ T_\sharp;\ {T_\natural}^*, \text{ and} \tag{45}$$

$$R;\ [\neg \mathsf{gd}.T] \sqsubseteq [\neg \mathsf{gd}.S];\ R. \tag{46}$$

Proof.

$$
\begin{array}{ll}
 & \mathsf{do}\ S\ \mathsf{od};\ R \\
= & \{\text{Lemma 31}\} \\
 & {S_\natural}^*;\ (S_\sharp;\ {S_\natural}^*)^\omega;\ [\neg \mathsf{gd}.S];\ R \\
= & \{\text{Lemma 25(33) and (35)}\} \\
 & {S_\natural}^*;\ ([\mathsf{gd}.S \wedge \mathsf{gd}.S_\sharp];\ S_\sharp;\ {S_\natural}^*)^\omega;\ [\neg \mathsf{gd}.S];\ R \\
= & \{\text{basic guard rules and Lemma 25(33)}\} \\
 & {S_\natural}^*;\ ([\mathsf{gd}.S];\ S_\sharp;\ {S_\natural}^*)^\omega;\ [\neg \mathsf{gd}.S];\ R
\end{array}
$$

$\sqsubseteq$ {assumptions (45) and (46) and Theorem 22}
  $S_\natural{}^*;\ R;\ ([\mathsf{gd}.T];\ T_\sharp;\ T_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.T]$
= {Lemma 25(33) and basic guard rules}
  $S_\natural{}^*;\ R;\ ([\mathsf{gd}.T \wedge \mathsf{gd}.T_\sharp];\ T_\sharp;\ T_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.T]$
= {Lemma 25(33) and (35))}
  $S_\natural{}^*;\ R;\ (T_\sharp;\ T_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.T]$
$\sqsubseteq$ {assumption (44)}
  $R;\ T_\natural{}^*;\ (T_\sharp;\ T_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.T]$
= {Lemma 31}
  $R;\ \mathsf{do}\ T\ \mathsf{od}$

□

An alternative general simulation rule can be constructed using Theorem (23). The third proof obligation is weaker than (46), however, we require the extra proof obligation (50).

Theorem 34 (alternative general simulation). *Given sublinear$^+$ expectation transformers R, $S = S_\sharp \sqcap S_\natural$ and $T = T_\sharp \sqcap T_\natural$ such that ${S_\natural}^\omega = {S_\natural}^*$ and ${T_\natural}^\omega = {T_\natural}^*$, we have that* $\mathsf{do}\ S\ \mathsf{od};\ R \sqsubseteq R;\ \mathsf{do}\ T\ \mathsf{od}$*, if*

$$S_\natural{}^*;\ R \sqsubseteq R;\ T_\natural{}^*, \tag{47}$$

$$\{\mathsf{gd}.S\};\ S_\sharp;\ S_\natural{}^*;\ R \sqsubseteq R;\ \{\mathsf{gd}.T\};\ T_\sharp;\ T_\natural{}^*, \tag{48}$$

$$R;\ [\neg\mathsf{gd}.T_\sharp] \sqsubseteq [\neg\mathsf{gd}.S];\ R, \textit{ and} \tag{49}$$

$$[\mathsf{gd}.S_\sharp \vee \neg\mathsf{gd}.S];\ R \sqsubseteq R;\ [\mathsf{gd}.T_\sharp \vee \neg\mathsf{gd}.T]. \tag{50}$$

Proof.

  $\mathsf{do}\ S\ \mathsf{od};\ R$
= {Lemma 31}
  $S_\natural{}^*;\ (S_\sharp;\ S_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.S];\ R$
= {Lemma 25 (33)}
  $S_\natural{}^*;\ ([\mathsf{gd}.S_\sharp];\ S_\sharp;\ S_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.S];\ R$
$\sqsubseteq$ {assumptions (48-50), Theorem 23 (see below for deferred justification)}
  $S_\natural{}^*;\ R;\ ([\mathsf{gd}.T_\sharp];\ T_\sharp;\ T_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.T]$
= {Lemma 25 (33)}
  $S_\natural{}^*;\ R;\ (T_\sharp;\ T_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.T]$
$\sqsubseteq$ {assumption (47)}
  $R;\ T_\natural{}^*;\ (T_\sharp;\ T_\natural{}^*)^\omega;\ [\neg\mathsf{gd}.T]$
= {Lemma 31}
  $R;\ \mathsf{do}\ T\ \mathsf{od}$

To see why Theorem 23 is applicable in the third proof step, observe that proof obligations (28) and (29) hold given $g1 = \mathsf{gd}.S_\sharp$, $g2 = \neg\mathsf{gd}.S$, $p1 = \mathsf{gd}.T_\sharp$ and $p2 = \neg\mathsf{gd}.T$. For example,

$g1 \Rightarrow \neg g2$
$\Leftrightarrow$ {definition of $g1$ and $g2$}
$\mathsf{gd}.S_\sharp \Rightarrow \neg(\neg \mathsf{gd}.S)$
$\Leftrightarrow$ {simplify}
$\mathsf{gd}.S_\sharp \Rightarrow \mathsf{gd}.S$
$\Leftrightarrow$ {definition of S, Lemma 25 (35)}
$\mathsf{gd}.S_\sharp \Rightarrow \mathsf{gd}.S_\sharp \vee \mathsf{gd}.S_\natural$
$\Leftrightarrow$ {simplify}
true.

The property $p1 \Rightarrow \neg p2$, may be proved in the same way. □

Our stuttering sensitive rules are more general than those verified by Back and von Wright using algebraic methods for standard action systems in [5, 24]: they are more general because they allow stuttering steps to be removed as well as added during refinement.

The basic cosimulation and simulation rules are special cases of the general rules: they are the same when $S_\natural$ and $T_\natural$ is chosen to be magic. In the next section we describe how the proof obligations of our general rules may be decomposed into simpler forms.

4.3 Common Special Cases of the Data Refinement Rules

We present and verify lemmas that have equivalent or stronger proof obligations than the general cosimulation and simulation lemmas, but may be simpler to use in practice. We demonstrate the use of one of these simpler rules using an example.

Cosimulation. Back and von Wright [1] observed that for conjunctive predicate transformers $S_\sharp$, $S_\natural$, $T_\sharp$, $T_\natural$, and R, if $T_\sharp = T_1 \sqcap ... \sqcap T_n$, and $S_\sharp \sqsubseteq S_1 \sqcap ... \sqcap S_n$, then condition (42) holds if for each $1 \leq i \leq n$,

$$R;\ S_i;\ {S_\natural}^* \sqsubseteq T_i;\ {T_\natural}^*;\ R.$$

When our programs are sublinear$^+$ expectation transformers instead of conjunctive predicate transformers, we may decompose condition (42) from the general cosimulation rule in the same way.

Lemma 35. *For probabilistic action systems* $\mathsf{do}\ S_\sharp \sqcap S_\natural\ \mathsf{od}$ *and* $\mathsf{do}\ T_\sharp \sqcap T_\natural\ \mathsf{od}$, *and sublinear*$^+$ *expectation transformer* R, *such that* $T_\sharp = T_1 \sqcap ... \sqcap T_n$, *and* $S_\sharp \sqsubseteq S_1 \sqcap ... \sqcap S_n$, *we have that*

$$R;\ S_\sharp;\ {S_\natural}^* \sqsubseteq T_\sharp;\ {T_\natural}^*;\ R$$

(condition (42)) holds, if for each $1 \leq i \leq n$,

$$R;\ S_i;\ {S_\natural}^* \sqsubseteq T_i;\ {T_\natural}^*;\ R$$

holds.

Proof. The proof of Back and von Wright [1] applies to expectation transformers:

$R;\ S_\sharp;\ S_\natural{}^*$
$\sqsubseteq$ {assumption on $S_\sharp$}
$R;\ (\sqcap i : [1, ..., n] \bullet S_i);\ S_\natural{}^*$
$\sqsubseteq$ {left distributivity and right sub-distributivity}
$(\sqcap i : [1, ..., n] \bullet R;\ S_i;\ S_\natural{}^*)$
$\sqsubseteq$ {assumption (35)}
$(\sqcap i : [1, ..., n] \bullet T_i;\ T_\natural{}^*;\ R)$
$=$ {left distributivity}
$(\sqcap i : [1, ..., n] \bullet T_i);\ T_\natural{}^*;\ R$
$=$ {assumption on $T_\sharp$}
$T_\sharp;\ T_\natural{}^*;\ R.$

□

The following cosimulation rule is weaker than our general rule, however its proof obligations may be simpler to use in practice. It is of a similar form to the weak cosimulation rule presented by Back and von Wright [1] for standard action systems with trace semantics.

Theorem 36 (weak cosimulation). *Given sublinear$^+$ expectation transformers R, $S = S_\sharp \sqcap S_\natural$ and $T = T_\sharp \sqcap T_\natural$ such that ${S_\natural}^\omega = {S_\natural}^*$ and ${T_\natural}^\omega = {T_\natural}^*$, we have that $R;\ \mathsf{do}\ S\ \mathsf{od} \sqsubseteq \mathsf{do}\ T\ \mathsf{od};\ R$, if R is continuous,*

$$R;\ S_\natural{}^* \sqsubseteq T_\natural;\ R, \tag{51}$$

$$R;\ S_\sharp;\ S_\natural{}^* \sqsubseteq T_\sharp;\ R, \text{ and} \tag{52}$$

$$R;\ [\neg \mathsf{gd}.S \vee \mathsf{fail}.(S_\sharp;\ S_\natural{}^*)] \sqsubseteq [\neg \mathsf{gd}.T];\ R. \tag{53}$$

Proof. Proof obligations (43) and (53) are identical, so we show that if (52) and (51) hold, then (41) and (42) hold.

1. (51) $\Rightarrow$ (41)
   This follows directly from Theorem 17.
2. (52) $\wedge$ (51) $\Rightarrow$ (42)
   $R;\ S_\sharp;\ S_\natural{}^*$
   $=$ {Lemma 12 (7)}
   $R;\ S_\sharp;\ S_\natural{}^*;\ S_\natural{}^*$
   $\sqsubseteq$ {assumption (52)}
   $T_\sharp;\ R;\ S_\natural{}^*$
   $\sqsubseteq$ {(using 41) (proved in part 1)}
   $T_\sharp;\ T_\natural{}^*;\ R$

□

In Theorem 36, the weak iteration construct may be eliminated from the left hand side of (52) and (51) by using Lemma 12 (5) to strengthen these conditions to

$$R;\ S_\natural \sqsubseteq T_\natural;\ R \text{ and} \tag{54}$$

$$R;\ S_\sharp;\ S_\natural \sqsubseteq T_\sharp;\ R. \tag{55}$$

Simulation. For the general cosimulation rule, it was simple to decompose proof obligation (42). The general simulation proof obligation (45) may be decomposed in a similar way, however it requires an extra proof obligation.

Theorem 37. *For probabilistic action systems* do S od *and* do T od*, and sublinear*$^+$ *expectation transformer* R*, where* $S = S_\sharp \sqcap S_\natural$*,* $T = T_\sharp \sqcap T_\natural$*,* $T_\sharp = T_1 \sqcap ... \sqcap T_n$*, and* $S_\sharp \sqsubseteq S_1 \sqcap ... \sqcap S_n$*, we have that*

$$\{\mathsf{gd}.S\};\ S_\sharp;\ {S_\natural}^*;\ R \sqsubseteq R;\ \{\mathsf{gd}.T\};\ T_\sharp;\ {T_\natural}^*$$

(condition (45)) holds, if for each $1 \leq i \leq n$*,*

$$\{\mathsf{gd}.S\};\ S_i;\ {S_\natural}^*;\ R \sqsubseteq R;\ \{\mathsf{gd}.T\};\ T_i;\ {T_\natural}^* \text{ and} \tag{56}$$

$$R;\ (\sqcap i : [1, ..., n] \bullet \{\mathsf{gd}.T\};\ T_i) = (\sqcap i : [1, .., n] \bullet R;\ \{\mathsf{gd}.T\};\ T_i). \tag{57}$$

Proof.

$\{\mathsf{gd}.S\};\ S_\sharp;\ {S_\natural}^*;\ R$
$\sqsubseteq$ {assumption on $S_\sharp$}
$\{\mathsf{gd}.S\};\ (\sqcap i : [1, ..., n] \bullet S_i);\ {S_\natural}^*;\ R$
$=$ {left distributivity and right distributivity of conjunctive programs}
$(\sqcap i : [1, ..., n] \bullet \{\mathsf{gd}.S\};\ S_i;\ {S_\natural}^*;\ R)$
$\sqsubseteq$ {assumption (56)}
$(\sqcap i : [1, ..., n] \bullet R;\ \{\mathsf{gd}.T\};\ T_i;\ {T_\natural}^*)$
$=$ {left distributivity}
$(\sqcap i : [1, ..., n] \bullet R;\ \{\mathsf{gd}.T\};\ T_i);\ {T_\natural}^*$
$=$ (assumption (57))
$R;\ (\sqcap i : [1, ..., n] \bullet \{\mathsf{gd}.T\};\ T_i);\ {T_\natural}^*$
$=$ {right distributivity of conjunctive programs}
$R;\ \{\mathsf{gd}.T\};\ (\sqcap i : [1, ..., n] \bullet T_i);\ {T_\natural}^*$
$=$ {assumption on $T_\sharp$}
$R;\ \{\mathsf{gd}.T\};\ T_\sharp;\ {T_\natural}^*$

□

Proof obligation (57) is trivially satisfied if R is conjunctive. The following rule is the simulation counterpart of our weak cosimulation rule.

Theorem 38 (weak simulation). *Given sublinear*$^+$ *expectation transformers* R*,* $S = S_\sharp \sqcap S_\natural$ *and* $T = T_\sharp \sqcap T_\natural$ *such that* ${S_\natural}^\omega = {S_\natural}^*$ *and* ${T_\natural}^\omega = {T_\natural}^*$*, we have that* do S od; $R \sqsubseteq R$; do T od*, if*

$${S_\natural}^*;\ R \sqsubseteq R;\ (T_\natural \sqcap \mathsf{skip}), \tag{58}$$

$$\{\mathsf{gd}.S\};\ S_\sharp;\ {S_\natural}^*;\ R \sqsubseteq R;\ \{\mathsf{gd}.T\};\ T_\sharp, \text{ and} \tag{59}$$

$$R;\ [\neg\mathsf{gd}.T] \sqsubseteq [\neg\mathsf{gd}.S];\ R. \tag{60}$$

Proof. Proof obligations (46) and (60) are identical, so we show that if (59) and (58) hold, then (44) and (45) hold.

1. (58) $\Rightarrow$ (44)
   This follows directly from cocontinuity of R and $T_\natural$ (from Theorem 2 we have that sublinear expectation transformers are cocontinuous), and Theorem 18.

2. (59) $\wedge$ (58) $\Rightarrow$ (45)

$$
\begin{array}{ll}
 & \{\mathsf{gd}.S\};\ S_\sharp;\ {S_\natural}^*;\ R \\
= & \{\text{Lemma 12 (7)}\} \\
 & \{\mathsf{gd}.S\};\ S_\sharp;\ {S_\natural}^*;\ {S_\natural}^*;\ R \\
\sqsubseteq & \{\text{using (44) (proved in part 1) }\} \\
 & \{\mathsf{gd}.S\};\ S_\sharp;\ {S_\natural}^*;\ R;\ {T_\natural}^* \\
\sqsubseteq & \{\text{assumption (59)}\} \\
 & R;\ \{\mathsf{gd}.T\};\ T_\sharp;\ {T_\natural}^*
\end{array}
$$

□

As for the weak cosimulation rule, the weak iteration construct may be eliminated from the left hand side of (59) and (58) using Lemma 12 (5) to strengthen these conditions to

$$S_\natural;\ R \sqsubseteq R;\ (T_\natural \sqcap \mathsf{skip}) \text{ and} \tag{61}$$

$$\{\mathsf{gd}.S\};\ S_\sharp;\ S_\natural;\ R \sqsubseteq R;\ \{\mathsf{gd}.T\};\ T_\sharp. \tag{62}$$

An equivalent theorem also exists for our alternative general simulation rule, Theorem 34. It may be verified in the same way as Theorem 38.

Theorem 39 (alternative weak simulation). *Given sublinear$^+$ expectation transformers R, $S = S_\sharp \sqcap S_\natural$ and $T = T_\sharp \sqcap T_\natural$ such that ${S_\natural}^\omega = {S_\natural}^*$ and ${T_\natural}^\omega = {T_\natural}^*$, we have that* $\mathsf{do}\ S\ \mathsf{od};\ R \sqsubseteq R;\ \mathsf{do}\ T\ \mathsf{od}$*, if*

$$S_\natural{}^*;\ R \sqsubseteq R;\ (T_\natural \sqcap \mathsf{skip}), \tag{63}$$

$$\{\mathsf{gd}.S\};\ S_\sharp;\ {S_\natural}^*;\ R \sqsubseteq R;\ \{\mathsf{gd}.T\};\ T_\sharp, \tag{64}$$

$$R;\ [\neg\mathsf{gd}.T_\sharp] \sqsubseteq [\neg\mathsf{gd}.S];\ R, \textit{ and} \tag{65}$$

$$[\mathsf{gd}.S_\sharp \vee \neg\mathsf{gd}.S];\ R \sqsubseteq R;\ [\mathsf{gd}.T_\sharp \vee \neg\mathsf{gd}.T]. \tag{66}$$

We present a simple example to demonstrate how the alternative weak simulation rule may be used in practice.

Example. Action system $S1$ (see Fig. 5) may be used to represent the behaviour of a unfair scheduler with two processes, $P1$ and $P2$, where both $P1$ and $P2$ are feasible. Predicates $env1$ and $env2$ indicate when processes $P1$ and $P2$ are able to be executed. If both processes are able to be executed at the same time, then the scheduler may demonically chose between executing $P1$ or $P2$, if only one process is ready, then it must execute that process, and if neither process is ready it terminates. We may show that this scheduler is data refined by action system $S2$ through representation program R. $S2$ represents a fair scheduler that has the same processes as $S1$, but, when both processes are able to be executed simultaneously, it chooses between them with equal probability. The fair scheduler, $S2$, uses fresh variable a to determine which process to execute. It also includes an extra action that is used to update the variable a. Fresh boolean variable c is used to specify when the new action should be executed.

We may use Theorem 39 to verify that $S1;\ R \sqsubseteq R;\ S2$ using the following partitioning of actions.

$$
\begin{array}{lll}
S1 & \triangleq & \mathsf{do}\ [env1];\ P1 \sqcap [env2];\ P2\ \mathsf{od} \\
S2 & \triangleq & \mathsf{do}\ [a = 1 \wedge c];\ P1;\ c := \mathsf{false} \\
 & & \quad \sqcap [a = 2 \wedge c];\ P2;\ c := \mathsf{false} \\
 & & \quad \sqcap [\neg c];\ c := \mathsf{true};\ C \\
 & & \quad \mathsf{od} \\
C & \triangleq & [env1 \wedge env2];\ (a := 1 \mathbin{_{\frac{1}{2}}\!\oplus} a := 2) \\
 & & \quad \sqcap [env1 \wedge \neg env2];\ a := 1 \\
 & & \quad \sqcap [\neg env1 \wedge env2];\ a := 2 \\
 & & \quad \sqcap [\neg env1 \wedge \neg env2];\ a := 0 \\
R & \triangleq & (c := \mathsf{true};\ C) \sqcap (c := \mathsf{false};\ a :\in \{0, 1, 2\})
\end{array}
$$

Fig. 5. Unfair and fair schedulers $S1$ and $S2$, and representation program R. We write $a :\in \{0, 1, 2\}$ to mean $a := 0 \sqcap a := 1 \sqcap a := 2$. Assignment statements have the usual meaning.

$$
\begin{array}{l}
S1_\sharp \triangleq [env1];\ P1 \sqcap [env2];\ P2 \\
S1_\natural \triangleq \mathsf{magic} \\
S2_\sharp \triangleq [a = 1 \wedge c];\ P1;\ c := \mathsf{false} \sqcap [a = 2 \wedge c];\ P2;\ c := \mathsf{false} \\
S2_\natural \triangleq [\neg c];\ c := \mathsf{true};\ C
\end{array}
$$

None of the actions from $S1$ are defined to be stuttering, however the action from $S2$ in which the next process is chosen is defined to be stuttering. $S1_\natural$ and $S2_\natural$ trivially satisfies the theorem conditions ${S1_\natural}^* = {S1_\natural}^\omega$ and ${S2_\natural}^* = {S2_\natural}^\omega$. We have that

$$
\begin{array}{l}
{S1_\natural}^* = \mathsf{skip} \text{ and} \\
{S2_\natural}^* = [\neg c];\ c := \mathsf{true};\ C \sqcap \mathsf{skip}.
\end{array}
$$

Proof of condition (63):

$$
\begin{array}{ll}
 & R;\ (S2_\natural \sqcap \mathsf{skip}) \\
= & \{\text{expanding } R \text{ and } S2_\natural, \text{ left distributivity }\} \\
 & c := \mathsf{true};\ C;\ ([\neg c];\ c := \mathsf{true};\ C \sqcap \mathsf{skip}) \\
 & \sqcap\ c := \mathsf{false};\ a :\in \{0, 1, 2\};\ ([\neg c];\ c := \mathsf{true};\ C \sqcap \mathsf{skip}) \\
= & \{C \text{ does not modify } c, \text{ right distributivity of conjunctive programs}\} \\
 & c := \mathsf{true};\ C;\ ([\mathsf{False}];\ c := \mathsf{true};\ C \sqcap \mathsf{skip}) \\
 & \sqcap\ c := \mathsf{false};\ a :\in \{0, 1, 2\};\ [\mathsf{True}];\ c := \mathsf{true};\ C \\
 & \sqcap\ c := \mathsf{false};\ a :\in \{0, 1, 2\};\ \mathsf{skip} \\
= & \{\text{preemption, } \mathsf{magic} \text{ is unit, } \mathsf{skip} \text{ is unit, definition of } C\} \\
 & c := \mathsf{true};\ C \sqcap c := \mathsf{true};\ C \sqcap c := \mathsf{false};\ a :\in \{0, 1, 2\} \\
= & \{\text{idempotence}\} \\
 & R \\
= & \{\text{definition of } S1_\natural, \mathsf{skip} \text{ is unit}\} \\
 & {S1_\natural}^*;\ R.
\end{array}
$$

Proof of condition (64): it is interesting to note that there is no suitable decomposition of actions $S1_\sharp$ and $S2_\sharp$ that enables us to use Theorem 37 to simplify

this proof step. Starting with the right hand side.

$\{\mathsf{gd}.S1\};\ S1_\sharp;\ S1_\natural{}^*;\ R$
$=$ {expanding definitions}
$\{env1 \vee env2\};\ ([env1];\ P1 \sqcap [env2];\ P2);\ (c := \mathsf{true};\ C \sqcap c := \mathsf{false};\ a :\in \{0,1,2\})$
$\sqsubseteq$ {left distributivity, basic lattice property}
$\{env1 \vee env2\};\ ([env1];\ P1;\ c := \mathsf{false};\ a := 1 \sqcap [env2];\ P2;\ c := \mathsf{false};\ a := 2)$
$=$ {basic guard property $\{p\} = [\neg p];\ \mathsf{abort} \sqcap [p]$}
$[env1 \vee env2];\ ([env1];\ P1;\ c := \mathsf{false};\ a := 1 \sqcap [env2];\ P2;\ c := \mathsf{false};\ a := 2)$
$\sqcap [\neg env1 \wedge \neg env2];\ \mathsf{abort}$
$=$ {basic guard rules, simplify}
$[env1 \wedge env2];\ (P1;\ c := \mathsf{false};\ a := 1 \sqcap P2;\ c := \mathsf{false};\ a := 2)$
$\sqcap [env1 \wedge \neg env2];\ P1;\ c := \mathsf{false};\ a := 1$
$\sqcap [\neg env1 \wedge env2];\ P2;\ c := \mathsf{false};\ a := 2$
$\sqcap [\neg env1 \wedge \neg env2];\ \mathsf{abort}$
$=$ {demonic choice refined by probabilistic choice}
$[env1 \wedge env2];\ (P1;\ c := \mathsf{false};\ a := 1_{\frac{1}{2}} \oplus P2;\ c := \mathsf{false};\ a := 2)$
$\sqcap [env1 \wedge \neg env2];\ P1;\ c := \mathsf{false};\ a := 1$
$\sqcap [\neg env1 \wedge env2];\ P2;\ c := \mathsf{false};\ a := 2$
$\sqcap [\neg env1 \wedge \neg env2];\ \mathsf{abort}$

Which we now show to be equivalent to $R;\ \{\mathsf{gd}.S2\};\ S2_\sharp$ by

$R;\ \{\mathsf{gd}.S2\};\ S2_\sharp$
$=$ {definition of $S2_\sharp$ and $S2$, basic guard rules}
$R;\ \{(c \wedge a \in \{1,2\}) \vee \neg c)\};\ [c \wedge a \in \{1,2\}];\ S2_\sharp$
$=$ {definition of R, left distributivity}
$(c := \mathsf{true};\ C;\ \{(c \wedge a \in \{1,2\}) \vee \neg c\};\ [c \wedge a \in \{1,2\}]$
$\sqcap c := \mathsf{false};\ a :\in \{0,1,2\};\ \{(c \wedge a \in \{1,2\}) \vee \neg c\};\ [c \wedge a \in \{1,2\}]);\ S2_\sharp$
$=$ {simplify}
$(c := \mathsf{true};\ C;\ \{a \in \{1,2\}\};\ [a \in \{1,2\}]$
$\sqcap c := \mathsf{false};\ a :\in \{0,1,2\};\ \{\mathsf{True}\};\ [\mathsf{False}]);\ S2_\sharp$
$=$ {basic guard rules, magic annihilates non-aborting statements }
$(c := \mathsf{true};\ C;\ \{a \in \{1,2\}\} \sqcap \mathsf{magic});\ S2_\sharp$
$=$ {magic is top}
$c := \mathsf{true};\ C;\ \{a \in \{1,2\}\};\ S2_\sharp$
$=$ {expanding C, left distributivity, simplify}
$c := \mathsf{true};$
$([env1 \wedge env2];\ (a := 1_{\frac{1}{2}} \oplus a := 2);\ \{\mathsf{True}\};\ S2_\sharp$
$\sqcap [env1 \wedge \neg env2];\ a := 1;\ \{\mathsf{True}\};\ S2_\sharp$
$\sqcap [\neg env1 \wedge env2];\ a := 2;\ \{\mathsf{True}\};\ S2_\sharp$
$\sqcap [\neg env1 \wedge \neg env2];\ a := 0;\ \{\mathsf{False}\};\ S2_\sharp)$
$=$ {abort is bottom and it annihilates feasible statements, skip is unit}
$c := \mathsf{true};$
$([env1 \wedge env2];\ (a := 1_{\frac{1}{2}} \oplus a := 2);\ S2_\sharp$
$\sqcap [env1 \wedge \neg env2];\ a := 1;\ S2_\sharp$
$\sqcap [\neg env1 \wedge env2];\ a := 2;\ S2_\sharp$
$\sqcap [\neg env1 \wedge \neg env2];\ \mathsf{abort}$

$=$ {expand $S2_\sharp$, left distributivity of probabilistic choice, magic is unit}
$\quad c := \mathsf{true};$
$\quad ([env1 \wedge env2];\ (a := 1;\ P1;\ c := \mathsf{false};\ _{\frac{1}{2}}\oplus\ a := 2;\ P2;\ c := \mathsf{false})$
$\quad \sqcap [env1 \wedge \neg env2];\ a := 1;\ P1;\ c := \mathsf{false}$
$\quad \sqcap [\neg env1 \wedge env2];\ a := 2;\ P2;\ c := \mathsf{false}$
$\quad \sqcap [\neg env1 \wedge \neg env2];\ \mathsf{abort}.$

Since a and c are not free in $env1$, $env2$, $P1$ and $P2$, the equivalence is established.

Proof of condition (65): Starting with the left hand side.

$\quad R;\ [\neg \mathsf{gd}.S2_\sharp]$
$=$ {definition of $S2_\sharp$}
$\quad R;\ [\neg c \vee a \notin \{1,2\}]$
$=$ {definition of R, left distributivity, simplify}
$\quad c := \mathsf{true};\ C;\ [a \notin \{1,2\}] \sqcap c := \mathsf{false};\ a :\in \{0,1,2\};\ [\mathsf{True}]$
$=$ {definition of C, left distributivity, simplify}
$\quad c := \mathsf{true};\ ([env1 \wedge env2];\ (a := 1\ _{\frac{1}{2}}\oplus\ a := 2);\ [\mathsf{False}]$
$\quad \sqcap [env1 \wedge \neg env2];\ a := 1;\ [\mathsf{False}]$
$\quad \sqcap [\neg env1 \wedge env2];\ a := 2;\ [\mathsf{False}]$
$\quad \sqcap [\neg env1 \wedge \neg env2];\ a := 0;\ [\mathsf{True}])$
$\quad \sqcap c := \mathsf{false};\ a :\in \{0,1,2\};\ [\mathsf{True}]$
$\sqsubseteq$ {magic is unit and annihilates non-aborting statements, skip is unit}
$\quad c := \mathsf{true};\ [\neg env1 \wedge \neg env2];\ a := 0 \sqcap c := \mathsf{false};\ a :\in \{0,1,2\}$
$\sqsubseteq$ {guards refine skip}
$\quad c := \mathsf{true};\ [\neg env1 \wedge \neg env2];\ a := 0$
$\quad \sqcap [\neg env1 \wedge \neg env2];\ c := \mathsf{false};\ a :\in \{0,1,2\}$
$\sqsubseteq$ {c not free in $env1$ or $env2$}
$\quad [\neg env1 \wedge \neg env2];\ c := \mathsf{true};\ a := 0$
$\quad \sqcap [\neg env1 \wedge \neg env2];\ c := \mathsf{false};\ a :\in \{0,1,2\}$

Which can be shown to equal the right hand side by

$\quad [\neg \mathsf{gd}.S1];\ R$
$=$ {definition of $S1$}
$\quad [\neg env1 \wedge \neg env2];\ R$
$=$ {guards are conjunctive, right distributivity of conjunctive programs}
$\quad [\neg env1 \wedge \neg env2];\ c := \mathsf{true};\ C$
$\quad \sqcap [\neg env1 \wedge \neg env2];\ c := \mathsf{false};\ a :\in \{0,1,2\}$
$=$ {definition of C, simplify}
$\quad [\neg env1 \wedge \neg env2];\ c := \mathsf{true};$
$\quad ([\mathsf{False}];\ (a := 1\ _{\frac{1}{2}}\oplus\ a := 2) \sqcap [\mathsf{False}];\ a := 1 \sqcap [\mathsf{False}];\ a := 2 \sqcap [\mathsf{True}];\ a := 0)$
$\quad \sqcap [\neg env1 \wedge \neg env2];\ c := \mathsf{false};\ a :\in \{0,1,2\}$
$=$ {preemption, magic is unit, skip is unit}
$\quad [\neg env1 \wedge \neg env2];\ c := \mathsf{true};\ a := 0$
$\quad \sqcap [\neg env1 \wedge \neg env2];\ c := \mathsf{false};\ a :\in \{0,1,2\}.$

Proof of condition (66):

$$
\begin{array}{ll}
 & [\mathsf{gd}.S1_\sharp \vee \neg \mathsf{gd}.S1];\ R \\
= & \{\text{definition of } S1_\sharp\} \\
 & [\mathsf{gd}.S1 \vee \neg \mathsf{gd}.S1];\ R \\
= & \{\text{simplify, } \mathsf{skip} \text{ is unit}\} \\
 & R \\
\sqsubseteq & \{\text{guards refine } \mathsf{skip}\} \\
 & R;\ [\mathsf{gd}.S2_\sharp \vee \neg \mathsf{gd}.S2].
\end{array}
$$

5 Loops

Probabilistic *while loops* are a generalisation of the standard while loops presented by Back and von Wright [5]. They are of a similar form to action systems. Their traditional expression in terms of the fixpoint operators is

$$
\begin{array}{l}
\mathsf{do}\ g_1 \rightarrow S_1 \ [\!]\ ...\ [\!]\ g_n \rightarrow S_n\ \mathsf{od} \triangleq \\
\qquad (\mu\, X \bullet [g_1];\ S_1;\ X \sqcap .. \sqcap [g_n];\ S_n;\ X \sqcap [\neg g1 \wedge ... \wedge \neg g_n]),
\end{array}
$$

which may be rewritten using the iteration constructs as

$$
\begin{array}{l}
\mathsf{do}\ g_1 \rightarrow S_1 \ [\!]\ ...\ [\!]\ g_n \rightarrow S_n\ \mathsf{od} = \\
\qquad ([g_1];\ S_1 \sqcap ... \sqcap [g_n];\ S_n)^\omega;\ [\neg g1 \wedge ... \wedge \neg g_n]).
\end{array}
$$

Probabilistic while loops differ from probabilistic action systems because guards are explicitly stated, and for each action $[g_i];\ S_i$, the guard of $[g_i];\ S_i$ need not equal g_i: that is, S_i may behave miraculously even when its guard g_i is enabled. (Recall from the previous section that the guards of actions were implicit).

We present two of the better known loop transformation rules: leapfrog and decomposition, as well as data refinement rules for loops.

5.1 Leapfrog and Decomposition for Loops

As for the other probabilistic leapfrog and decomposition rules, we have that for monotonic expectation transformers, we only get a weaker form of the leapfrog rule, but for decomposition, the rule is exactly the same as for standard loops (we do not require conjunctivity as required in [5]).

Lemma 40 (loop leapfrog). *Given monotonic expectation transformers S and T, if*

$$S;\ [\neg h] \sqsubseteq [\neg g];\ S \text{ and } S;\ [h] \sqsubseteq [g];\ S \tag{67}$$

then

$$S;\ \mathsf{do}\ h \rightarrow T;\ S\ \mathsf{od} \sqsubseteq \mathsf{do}\ g \rightarrow S;\ T\ \mathsf{od};\ S. \tag{68}$$

For conjunctive predicate transformers, Back and von Wright [5] specified the assumptions for the leapfrog rule as

$$S.\mathsf{True} \wedge \neg g \leq S.(\neg h) \text{ and } S.\mathsf{True} \wedge g \leq S.h. \tag{69}$$

The guard propagation rules for conjunctive predicate transformers can be used to show that assumptions (67) and (69) are equivalent. Since the same guard propagation rules do not in general apply to our expectation transformers, we may not express our constraints in this way.

Proof. The proof differs from that of Back and von Wright [5] because in the second step we have refinement instead of equality since we cannot assume conjunctivity of S.

$$\begin{array}{ll} & S;\ \mathsf{do}\ h \to T;\ S\ \mathsf{od} \\ = & \{\text{rewrite loop using iteration constructs}\} \\ & S;\ ([h];\ T;\ S)^\omega;\ [\neg h] \\ \sqsubseteq & \{\text{sub-leapfrog (Theorem 14)}\} \\ & (S;\ [h];\ T)^\omega;\ S;\ [\neg h] \\ \sqsubseteq & \{\text{assumptions } S;\ [\neg h] \sqsubseteq [\neg g];\ S \text{ and } S;\ [h] \sqsubseteq [g];\ S\ \} \\ & ([g];\ S;\ T)^\omega;\ [\neg g];\ S \\ = & \{\text{rewrite iteration in terms of loop}\} \\ & \mathsf{do}\ g \to S;\ T\ \mathsf{od};\ ;\ S \end{array}$$

□

Lemma 41 (loop decomposition). *Given monotonic expectation transformers S and T,*

$$\begin{array}{l} \mathsf{do}\ g \wedge \neg h \to S \parallel h \to T\ \mathsf{od} \\ \qquad = \mathsf{do}\ h \to T\ \mathsf{od};\ \mathsf{do}\ g \to (S;\ \mathsf{do}\ h \to T\ \mathsf{od})\ \mathsf{od}. \end{array}$$

Proof. The proof of this is the same as that presented by Back and von Wright [5]. We present it for clarity. We emphasise here that we may use the leapfrog rule in the second step since guards are conjunctive, not because we have assumed either S or T are conjunctive.

$$\begin{array}{ll} & \mathsf{do}\ h \to T\ \mathsf{od};\ \mathsf{do}\ g \to (S;\ \mathsf{do}\ h \to T\ \mathsf{od})\ \mathsf{od} \\ = & \{\text{rewrite loops using iteration constructs}\} \\ & ([h];\ T)^\omega;\ [\neg h];\ ([g];\ S;\ ([h];\ T)^\omega;\ [\neg h])^\omega;\ [\neg g] \\ = & \{\text{coercions are conjunctive, leapfrog (Theorem 14), basic guard rules}\} \\ & ([h];\ T)^\omega;\ ([g \wedge \neg h];\ S;\ ([h];\ T)^\omega)^\omega;\ [\neg g \wedge \neg h] \\ = & \{\text{decomposition (Lemma 13)}\} \\ & ([g \wedge \neg h];\ S \sqcap [h];\ T)^\omega;\ [\neg g \wedge \neg h] \\ = & \{\text{rewrite iteration in terms of loop}\} \\ & \mathsf{do}\ g \wedge \neg h \to S \parallel h \to T\ \mathsf{od} \end{array}$$

□

5.2 Data Refinement of Loops

The data refinement rules for loops are similar for those for action systems. We present the basic and general loop simulation and cosimulation rules, however we elide proofs because of their similarity to those in Sect. 4.

Lemma 42 (basic loop cosimulation). *Given monotonic expectation transformers R, S and T, and predicates g and p, we have that* $R;\ \mathsf{do}\ g \to S\ \mathsf{od} \sqsubseteq \mathsf{do}\ p \to T\ \mathsf{od};\ R$, *if R is continuous,*

$$R;\ [g];\ S \sqsubseteq [p];\ T;\ R, \text{ and} \tag{70}$$

$$R;\ [\neg g \vee \mathsf{fail}.S] \sqsubseteq [\neg p];\ R. \tag{71}$$

Lemma 43 (basic loop simulation). *Given monotonic expectation transformers R, S and T, and predicates g and p, we have that* $\mathsf{do}\ g \to S\ \mathsf{od};\ R \sqsubseteq R;\ \mathsf{do}\ p \to T\ \mathsf{od}$, *if*

$$\{g\};\ S;\ R \sqsubseteq R;\ \{p\};\ T, \text{ and} \tag{72}$$

$$R;\ [\neg p] \sqsubseteq [\neg g];\ R. \tag{73}$$

As for the action system basic data refinement rules, these rules may be verified using commutativity Theorems 21 and 22.

Lemma 44 (general loop cosimulation). *Given predicates $g1$, $g2$, $p1$, $p2$, and monotonic expectation transformers R, $S_\sharp$, $S_\natural$ and $T_\sharp$, $T_\natural$, such that $([g2];\ S_\natural)^\omega = ([g2];\ S_\natural)^*$ and $([p2];\ T_\natural)^\omega = ([p2];\ T_\natural)^*$, we have that*

$$R;\ \mathsf{do}\ g1 \to S_\sharp \parallel g2 \to S_\natural\ \mathsf{od} \sqsubseteq \mathsf{do}\ p1 \to T_\sharp \parallel p2 \to T_\natural\ \mathsf{od};\ R,$$

provided R is continuous,

$$R;\ ([g2];\ S_\natural)^* \sqsubseteq ([p2];\ T_\natural)^*;\ R, \tag{74}$$

$$R;\ [g1];\ S_\sharp;\ ([g2];\ S_\natural)^* \sqsubseteq [p1];\ T_\sharp;\ ([p2];\ T_\natural)^*;\ R, \text{ and} \tag{75}$$

$$R;\ [(\neg g1 \wedge \neg g2) \vee \mathsf{fail}.([g1];\ S_\sharp;\ ([g2];\ S_\natural)^*)] \sqsubseteq [\neg p1 \wedge \neg p2];\ R. \tag{76}$$

Lemma 45 (general loop simulation). *Given predicates $g1$, $g2$, $p1$, $p2$, monotonic expectation transformers R, $S_\sharp$, $S_\natural$ and $T_\sharp$, $T_\natural$ such that $([g2];\ S_\natural)^\omega = ([g2];\ S_\natural)^*$ and $([p2];\ T_\natural)^\omega = ([p2];\ T_\natural)^*$, we have that*

$$\mathsf{do}\ g1 \to S_\sharp \parallel g2 \to S_\natural\ \mathsf{od};\ R \sqsubseteq R;\ \mathsf{do}\ p1 \to T_\sharp \parallel p2 \to T_\natural\ \mathsf{od},$$

provided

$$([g2];\ S_\natural)^*;\ R \sqsubseteq R;\ ([p2];\ T_\natural)^*, \tag{77}$$

$$\{g1 \vee g2\};\ [g1];\ S_\sharp;\ ([g2];\ S_\natural)^*;\ R \sqsubseteq R;\ \{p1 \vee p2\};\ [p1];\ T_\sharp;\ ([p2];\ T_\natural)^*, \text{ and} \tag{78}$$

$$R;\ [\neg p1 \wedge \neg p2] \sqsubseteq [\neg g1 \wedge \neg g2];\ R. \tag{79}$$

Lemma 46 (alternative general loop simulation). *Given predicates $g1$, $g2$, $p1$, $p2$, monotonic expectation transformers R, $S_\sharp$, $S_\natural$ and $T_\sharp$, $T_\natural$ such that $([g2];\ S_\natural)^\omega = ([g2];\ S_\natural)^*$ and $([p2];\ T_\natural)^\omega = ([p2];\ T_\natural)^*$, we have that*

$$\mathsf{do}\ g1 \to S_\sharp \parallel g2 \to S_\natural\ \mathsf{od};\ R \sqsubseteq R;\ \mathsf{do}\ p1 \to T_\sharp \parallel p2 \to T_\natural\ \mathsf{od},$$

provided

$$([g2];\ S_\natural)^*;\ R \sqsubseteq R;\ ([p2];\ T_\natural)^*, \tag{80}$$

$$\{g1 \vee g2\};\ [g1];\ S_\sharp;\ ([g2];\ S_\natural)^*;\ R \sqsubseteq R;\ \{p1 \vee p2\};\ [p1];\ T_\sharp;\ ([p2];\ T_\natural)^*, \tag{81}$$

$$R;\ [\neg p1] \sqsubseteq [\neg g1 \wedge \neg g2];\ R, \text{ and} \tag{82}$$

$$[g1 \vee \neg g2];\ R \sqsubseteq R;\ [p1 \vee \neg p2]. \tag{83}$$

Weaker versions of these rules may be generated for loops in the same way as we did for action systems in Sect. 4.

6 Conclusion

Back and von Wright have demonstrated how to reason about standard loops in a concrete algebraic setting [5, 2]. We have demonstrated how probabilistic loops may be reasoned about in a similar way. We have identified a number of important transformation rules that are common to both probabilistic and standard loops. In addition, we have identified a number of standard transformation rules that are not applicable to probabilistic programs. For the latter rules, we have developed alternative transformation rules that are suitable in the probabilistic context. We have applied our algebraic rules to develop transformation rules for probabilistic action systems and while-loops. In particular, we have constructed new data refinement rules for probabilistic action systems and probabilistic while-loops. We have also analysed how our data refinement rules may be decomposed into simpler forms.

There are many benefits to taking an algebraic approach to reasoning about iterations and loops: the main benefit being that it can simplify reasoning about complex theorems. The transformation rules that we have developed may be used as a basis to develop further rules.

Acknowledgments. This research was supported by Australian Research Council (ARC) Discovery Grant DP0558408, *Analysing and generating fault-tolerant real-time systems.*

References

1. Ralph-Johan Back and Joakim von Wright. Trace refinement of action systems. In *International Conference on Concurrency Theory*, volume 836 of *LNCS*, pages 367–384. Springer Verlag, 1994.
2. Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.
3. R.J.R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *Proc. of the 2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 131–142. ACM Press, 1983.
4. R.J.R. Back and R. Kurki-Suonio. Distributed cooperation with action systems. *ACM Trans. Program. Lang. Syst.*, 10(4):513–554, 1988.
5. R.J.R Back and J. von Wright. Reasoning algebraically about loops. *Acta Informatica*, 36:295–334, 1999.
6. Ernie Cohen. Hypotheses in Kleene algebra. Technical Report TM-ARH-023814, Belcore, 1994.
7. Ernie Cohen. Separation and reduction. In *Mathematics of Program Construction*, volume 1837 of *LNCS*, pages 45–59. Springer, 2000.
8. B. A. Davey and H.A Priestley. *Introduction to Lattices*. Cambridge University Press, 1990.
9. Edsger W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.
10. J. He, K. Seidel, and A. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2-3):171–192, 1997.
11. Joe Hurd. A formal approach to probabilistic termination. In *TPHOLs*, volume 2410 of *LNCS*, pages 230–245. Springer-Verlag, 2002.

12. Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.
13. Annabelle McIver and Carroll Morgan. Demonic, angelic and unbounded probabilistic choices in sequential programs. *Acta Informatica*, 37(4/5):329–354, 2001.
14. Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.
15. Larissa Meinicke and Ian J. Hayes. Reasoning algebraically about probabilistic loops. In *Eighth Internatinal Conference on Formal Engineering Methods*, volume 4260 of *LNCS*, pages 380–399, 2006. Accepted for publication.
16. Bernhard Möller. Lazy Kleene algebra. In *Mathematics of Program Construction*, volume 3125 of *LNCS*, pages 252–273. Springer-Verlag, 2004.
17. C. Morgan. *Programming from Specifications*. Prentice Hall, second edition, 1994.
18. Carroll Morgan. Proof rules for probabilistic loops. In He Jifeng, John Cooke, and Peter Wallis, editors, *BCS-FACS 7th Refinement Workshop*. Springer Verlag, August 1996.
19. Carroll Morgan and Annabelle McIver. Cost analysis of games using program logic, 2001. Presented at 8th Asia-Pacific Software Engineering Conference.
20. Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 8(1):325–353, January 1999.
21. Kaisa Sere and Elena Troubitsyna. Probabilities in action systems. In *Proc. of the 8th Nordic Workshop on Programming Theory*, 1996.
22. Kim Solin and Joakim von Wright. Refinement algebra with operators for enabledness and termination. In *Mathematics of Program Construction*, volume 4014 of *LNCS*, pages 397–415, 2006.
23. Elena A. Troubitsyna. Reliability assessment through probabilistic refinement. *Nordic Journal of Computing*, pages 320–342, 1999.
24. Joakim von Wright. From Kleene algebra to refinement algebra. In *Mathematics of Program Construction*, volume 2386 of *LNCS*, pages 233–262. Springer, 2002.
25. Joakim von Wright. Towards a refinement algebra. *Science of Computer Programming*, 51, 2004.