Towards automated proof support for probabilistic distributed systems

AK McIver[1] and T Weber[2]

[1] Dept. Computer Science, Macquarie University, NSW 2109 Australia;
anabel@ics.mq.edu.au
[2] Dept. Computer Science, Technische Universität München, Germany;
webertj@in.tum.de

Abstract. The mechanisation of proofs for probabilistic systems is particularly challenging due to the verification of real-valued properties that probability entails: experience indicates [12, 4, 11] that there are many difficulties in automating real-number arithmetic in the context of other program features.
In this paper we propose a framework for verification of probabilistic distributed systems based on the generalisation of Kleene algebra with tests that has been used as a basis for development of concurrency control in standard programming [7]. We show that verification of real-valued properties in these systems can be considerably simplified, and moreover that there is an interpretation which is susceptible to counterexample search via state exploration, despite the underlying real-number domain.

1 Introduction

Recent developments in mechanised theorem-proving approaches to the verification of probabilistic programs [12, 4, 11] have highlighted at once both the benefits and drawbacks of proof-based techniques. On the plus side, we see clearly that proofs provide more general solutions (which can be re-used and easily checked) than do other forms of automated verification such as probabilistic model checking [17] (which, unlike standard model checking, does not compute counterexamples). On the minus side, however, experience has shown that there remain difficulties in automating real arithmetic in the context of other program features, a necessity whenever typical properties are quantitative such as "the probability that the program terminates is at least 0.7".

The infinite domain (of reals) needed for quantitative analysis also implies other limitations when compared to qualitative analysis. For example, it prevents the provision of counterexample search via state exploration [28]. Counterexample search is an effective technique in the activity of mathematical proof, as it leads to the debugging of conjectures, often by directing the prover to strengthen the hypotheses under which the consequent should follow. In program verification this debugging process corresponds to reformulations of specifications under which a proposed refinement is valid, or to the redesign of a suggested implementation. Such debugging strategies have been employed to great effect in tools

such as Alloy [13]. Given the above however, it appears at first sight that the task of quantitative verification cannot be enhanced straightforwardly by counterexample search.

Nevertheless, the benefits of mechanised proof provide a strong motivation to look for methods that will alleviate the drawbacks. In this paper we present a proof system which reduces the overhead of arithmetic in automated proof, as well as supporting counterexample search. As far as we are aware, this is the first proof system for probabilistic programs to do so.

As our principal context we take probabilistic distributed systems — such systems are particularly difficult to verify, as the interaction of probability with other system features can lead to unexpected behaviour [26, 2]. In these cases, much of the verification is devoted to showing that, under certain simple hypotheses, a highly distributed architecture is equivalent to a serialised one. In fact a proof of such an equivalence can often be done without appeal to probability at all, even though the systems and their correctness depend on explicit quantitative properties. This has the effect of reducing any quantitative reasoning to validation of the hypotheses for a particular concrete system, in general a significantly simpler problem than analysing the two architectures directly. Cohen's work [7, 6] on the practical use of *Kleene algebra* for verification in loosely-coupled (but non-probabilistic) systems provides ample evidence that a Kleene-style algebra is a good candidate for this kind of reasoning, and the evidence applies even when probability is introduced.

Our first contribution is to show how a model for probabilistic systems $\mathcal{LS}$ (for given state space S) [10, 23] can be interpreted over a Kleene-style program algebra [16], so that explicit probabilistic reasoning is significantly reduced.

Next we turn to the feasibility of counterexample search in probabilistic Kleene algebra. For this we propose an abstraction of the probabilistic model that preserves the important limiting features of standard probability, and at the same time yields genuinely finite models which are thus amenable to state exploration techniques similar to those for non-probabilistic systems [28].

Our second contribution is a model of abstract probabilities $\mathcal{KS}$ that is susceptible to complete semantic exploration, yielding counterexamples even for the probabilistic model $\mathcal{LS}$. This appears to be the first facility for counterexample search for probabilistic systems.

In Sec. 2 we set out the probabilistic model $\mathcal{LS}$, together with, in Sec. 2.1, an interpretation in the Kleene-style algebra. Next, in Sec. 3 we set out the abstract model $\mathcal{KS}$ and show that the the interpretation in Kleene algebra is a homomorphic image of the interpretation in $\mathcal{LS}$, and in particular that equalities over Kleene algebra expressions are preserved. Finally in Sec. 4 we show how state exploration techniques within $\mathcal{KS}$ can be used to generate counterexamples even within $\mathcal{LS}$. In Sec. 5 we summarise other research in this area and suggest further topics for investigation.

The notational conventions used are as follows. Function application is represented by a dot, as in $f.x$. If K is a set then $\overline{K}$ is the set of discrete probability distributions over K, that is the normalised functions from K into the real interval $[0,1]$ (*i.e.* function f is normalised if $\sum_{s:K} f.s = 1$). A point distribution centered at a point k is denoted by δ_k. The $(p, 1-p)$-weighted average of distributions d and d' is denoted $d\,{}_p\!\oplus d'$; more generally we write $p_1 d^1 + \ldots + p_n d^n$ for the $(p_1, \ldots, p_n)$ weighted average over distributions $d^1, \ldots, d^n$. If K is a subset, and d a distribution, we write $d.K$ for $\sum_{s:K} d.s$. The power set of K is denoted $\wp K$.

2 Probabilistic systems

Given a (discrete) state space S, the set of functions $S \to \wp\overline{S}$, from (initial) states to subsets of distributions over (final) states, is the basis for the transition-system style model now generally accepted for probabilistic systems [19] though, depending on the particular application, the conditions imposed on the subsets of (final) probability distributions can vary [23, 10]. Briefly the idea is that probabilistic systems comprise both *quantifiable* arbitrary behaviour (such as the chance of winning an automated lottery) together with *un*quantifiable arbitrary behaviour (such as the precise order of concurrent events). The functions $S \to \wp\overline{S}$ model the quantifiable events as probability distributions — effectively probabilistic transitions (hence the range of semantic functions includes distributions in $\overline{S}$). On the other hand, the unquantifiable events are modelled as a subset of distributions (hence the range of semantic functions can be a subset of distributions).

For example, a program that simulates a fair coin is modelled by a function that maps an arbitrary state s to the distribution weighted evenly between the point distributions representing heads and tails (but see below):

$$s \mapsto \{\delta_{\mathsf{head}}\ {}_{1/2}\!\oplus\ \delta_{\mathsf{tail}}\} \ . \tag{1}$$

In contrast a program that simulates a possible bias favouring heads of at most 2/3, is modelled by a function which takes an arbitrary state to a subset of distributions specifying the precise limits on the bias:

$$s \mapsto \{\delta_{\mathsf{head}}\ {}_{1/2}\!\oplus\ \delta_{\mathsf{tail}}\ , \delta_{\mathsf{head}}\ {}_{2/3}\!\oplus\ \delta_{\mathsf{tail}}\} \ . \tag{2}$$

In setting up the details, we follow Morgan *et al.*[23] and take a domain theoretical approach, restricting the result sets of the semantic functions according to an underlying order on the state space. An innovation, however, is to distinguish specially "miraculous" or infeasible behaviour from ordinary behaviour — miracles are used in program semantics to simplify calculations [21, 20], to model "tests" [16] and, here, they will work well with our simple algebra of programs to come. In the semantics, miracles will be associated with a special introduced state $\top$, and our program model is defined over the *probabilistic power domain* [14] based on the underlying (flat) domain $(S^\top, \sqsubseteq)$, where $S^\top$ is S conjoined

with the special state $\top$, and the order $\sqsubseteq$ is constructed so that $\top$ dominates all (proper) states in S, which are otherwise unrelated.

Definition 1. *A probabilistic power domain is a pair* $(\overline{S^\top}, \sqsubseteq_{\mathcal{D}})$*, where* $\overline{S^\top}$ *is the set of normalised functions from* $S^\top$ *into the real interval* $[0,1]$*, and* $\sqsubseteq_{\mathcal{D}}$ *is induced from* $\sqsubseteq$ *on* $S^\top$ *so that*

$$d \sqsubseteq_{\mathcal{D}} d' \quad \textit{iff} \quad (\forall K \subseteq S \cdot d.K + d.\top \quad \leq \quad d'.K + d'.\top) \ .$$

Probabilistic programs (with miracles) are now modelled as the set of functions from initial $S^\top$ to sets of final distributions over $S^\top$, where the result sets are restricted by so-called *healthiness conditions* characterising viable probabilistic behaviour, and motivated in detail elsewhere [19]. By doing so the semantics accounts for specific features of probabilistic programs. In this case (again following Morgan) we impose *up-closure* (the inclusion of all $\sqsubseteq_{\mathcal{D}}$-dominating distributions), *convex closure* (the inclusion of all convex combinations of distributions), and *Cauchy closure* (the inclusion of all limits of distributions according to the standard Cauchy metric on real-valued functions [23]). Thus, by construction, viable computations are those in which miracles dominate (refine) all other behaviours (implied by up-closure), nondeterministic choice is refined by probabilistic choice (implied by convex closure), and classic limiting behaviour of probabilistic events (such as so-called "zero-one laws" [3]) is also accounted for (implied by Cauchy closure). An additional bonus is that program refinement is simply defined as reverse set-inclusion. We observe that probabilistic properties are preserved with increasing order.

Definition 2. *The space of probabilistic programs* [4] *is given by* $(\mathcal{L}S, \sqsubseteq_{\mathcal{L}})$ *where* $\mathcal{L}S$ *is the set of functions from* $S^\top$ *to the power set of* $\overline{S^\top}$*, restricted to subsets which are* Cauchy- , convex- *and* up *closed with respect to* $\sqsubseteq_{\mathcal{D}}$*. All programs are* $\top$*-preserving (mapping* $\top$ *to* $\{\delta_\top\}$*). The order between programs is defined*

$$P \sqsubseteq_{\mathcal{L}} P' \quad \textit{iff} \quad (\forall s\colon S \cdot P.s \supseteq P'.s) \ .$$

Thus in the examples above, taking the closure conditions into account, we see that up-closure implies that the result set at (1) would also contain the distributions $a\delta_{\mathsf{head}} + b\delta_{\mathsf{tail}} + c\delta_\top$ for a, b, c satisfying the conditions $1/2 \leq a + c$, and $1/2 \leq b + c$. Similarly convex-closure implies that the result set at (2) should also include all $(p, 1-p)$-weighted distributions of the form $(\delta_{\mathsf{head}\ 2/3} \oplus \delta_{\mathsf{tail}}/3)\ {}_p\oplus$ $(\delta_{\mathsf{head}\ 1/2} \oplus \delta_{\mathsf{tail}})$, for any $0 \leq p \leq 1$.

In Fig. 1 we define some mathematical operators on the space of programs, which will be used to interpret our language. Informally composition $P; P'$ corresponds to a program P being executed followed by P', so that from initial state s, any result distribution d of $P.s$ can be followed by an arbitrary distribution of

[3] An easy consequence of a zero-one law is that if a fair coin is flipped repeatedly, then with probability 1 a head is observed eventually. See the program 'coin' inside an iteration, which is discussed below.

[4] This particular "Lamington" model was first suggested by Carroll Morgan [22].

Identity	$Id.s$	$\hat{=}$	$\lceil\{\delta_s\}\rceil$,
top	$\top.s$	$\hat{=}$	$\{\delta_\top\}$,
composition	$(P;P').s$	$\hat{=}$	$\{\sum_{u:S^\top}(d.u)\times d'_u \mid d\in P.s; d'_u\in P'.u\}$,
choice	$(\textit{if } B \textit{ then } P \textit{ else } P').s$	$\hat{=}$	$\textit{if } B.s, \textit{ then } P.s, \textit{ otherwise } P'.s$
probability	$(P\ {}_p\!\oplus P').s$	$\hat{=}$	$\lceil\{d\ {}_p\!\oplus d' \mid d\in r.s; d'\in r'.s\}\rceil$,
nondeterminism	$(P\sqcap P').s$	$\hat{=}$	$\lceil\{d \mid d\in(P.s\cup P'.s)\}\rceil$,
iteration	P^*	$\hat{=}$	$(\nu X\cdot P; X\sqcap Id)$.

In the above definitions s is a state in S and $\lceil K\rceil$ is the smallest up-, convex- and Cauchy-closed subset of distributions containing K. Programs are denoted by P and P', and the expression $(\nu X\cdot f.X)$ denotes the greatest fixed point of the function f — in the case of iteration the function is the monotone $\sqsubseteq_{\mathcal{L}}$-program-to-program function $\lambda X\cdot(P; X\ \sqcap\ Id)$. All programs map $\top$ to $\{\delta_\top\}$.

Fig. 1. Mathematical operators on the space of programs [19].

P'. [5] The probabilistic operator takes the weighted average of the distributions of its operands, and the nondeterminism operator takes their union. To illustrate, let S be the set of integers, and consider the following transition $\pi_k.s \mathrel{\hat{=}} \{\delta_{s+k}\}$. Thus $\pi_k.s$ is the transition that adds k to s. Next we can define more complicated transitions using the operators, for example

$$\Pi.s \quad \hat{=} \quad \textit{if } (s\geq 0) \textit{ then } (\pi_{-1}\ {}_{\frac{1}{2}}\!\oplus (Id\ {}_{\frac{1}{2}}\!\oplus \pi_{-2})).s \quad \textit{else } Id.s \tag{3}$$

The transition at (3) essentially maps initial states with value at least 0 to the weighted sum of point distributions, namely $\frac{1}{2}\times\delta_{s-1}+\frac{1}{4}\times\delta_s+\frac{1}{4}\times\delta_{s-2}$, and otherwise leaves the state alone.

Iteration is the most intricate of the operations — operationally P^* represents the program that can execute P an arbitrary number of finite times. In the probabilistic context, as well as generating the results of all "finite iterations" of $(P\sqcap Id)$ (*viz*, a finite number of compositions of $(P\sqcap Id)$), imposition of Cauchy closure acts as expected on metric spaces, in that it generates all *limiting* distributions as well — *i.e.* if $d_0,\ d_1,\ldots$ are distributions contained in a result set M which converge to d, then d is contained in M as well. To illustrate, consider the transition at (3) inside an iteration Π^* corresponding to a transition system which can (but does not have to) reduce s indefinitely until its value falls below zero. For states with value less than zero the iteration does nothing to the state (*i.e.* we are considering "skipping forever" to be the the same as terminating). Now it is easy to see that from initial $s=0$, after n iterations of the program at (3) the distribution over the results $s=0,-1$ or -2 is $p_n\delta_0+q_n\delta_{-1}+r_n\delta_{-2}$, where $p_n=1/(4^n)$, $q_n=2(1-p_n)/3$ and $r_n=(1-p_n)/3$. But observe now the limits $\lim_{n\to\infty}p_n=0$, $\lim_{n\to\infty}q_n=2/3$, and $\lim_{n\to\infty}r_n=1/3$, and that Cauchy closure implies the limit distribution $2\delta_{-1}/3+\delta_{-2}/3$ is contained in the result set of the iteration Π^* as well.

[5] Compare composition in Markov Decision Processes [9].

More generally we will need to characterise the limiting distributions in terms of distributions contained in the result sets of *finite iterations*. (Recall that a finite iteration of program Q is of the form Q^n for some n, where $Q^0 \mathrel{\hat{=}} Id$, and $Q^{n+1} \mathrel{\hat{=}} Q; Q^n$.) The following lemma provides conditions, in the context of finite state spaces, that a distribution be generated by an iteration. For any distribution d, let $\mathsf{supp.d}$ be the smallest subset $K \subseteq S^\top$ with $d.K = 1$, and for iteration (P^*) let $P^{\mathcal{N}}$ be the set of distributions generated by finite iterations of $(Id \sqcap P)$. Further, we say that "subset K can be reached with probability 1 from state s via executions of program Q" if there exists a sequence (possibly finite) of distributions d_i with each $d_i \in Q^i.s$, such that $\lim_{i\geq 0} d_i.K = 1$. (Note that distributions contained in finite executions are a special case.)

Lemma 1. *Let P be a program in $\mathcal{LS}$, and let S be finite. If distribution d is in $P^*.s$, then* $\mathsf{supp.d}$ *can be reached from s with probability* 1 *via executions of $(Id \sqcap P)$. Alternatively if K is a subset of* $\mathsf{supp.d'}$ *for some d' in $P^*.s$, and can be reached with probability* 1 *via executions of $(Id \sqcap P)$ from all s in* $\mathsf{supp.d'}$*, then there is some d in $P^*.s$ with* $\mathsf{supp.d} = \mathsf{K}$.

Proof. By the greatest fixed point definition, $P^* = \lceil \lim_{n\geq 0}(Id \sqcap P)^n \rceil$, thus the first condition follows. Alternatively if distribution d' is in the result set of $P^*.s$, then the nondeterminism in subsequent executions of $(Id \sqcap P)$ can be exploited to produce a distribution with support K, since if $s' \notin K$, the branch P can be selected until K is established with probability 1*, which (if P itself has no nondeterminsim) yields an appropriate distribution d. The case that P is nondeterministic reduces to the latter case since it has been shown elsewhere [19, 8] that $P^* = \sqcap_{i:I} P_i^*$, where we are using $\sqcap_{i:I} P_i^*$ to mean the nondeterministic choice over the programs in the index set I, which in this case ranges over all deterministic refinements of P. The result now follows since d is a convex combination of distributions within the $P_i^*.s$.*

Now we have introduced a model for general probabilistic contexts, our next task is to investigate its program algebra. That is the topic of the next section.

2.1 Kleene algebra for probabilistic systems

Kleene algebra consists of a sequential composition operator (with a distinguished identity (1) and zero (0)); a binary plus (+) and unary star ($*$). Terms are ordered by $\leq$ defined by $+$ (see Fig. 2), and both binary as well as the unary operators are monotone with respect to it. Sequential composition is indicated by the sequencing of terms in an expression so that ab means the program denoted by a is executed first, and then b. The expression $a + b$ means that either a or b is executed, and the Kleene star a^* represents an arbitrary number of executions of the program a. In Fig. 2 we set out the rules for the *probabilistic Kleene algebra*, *pKA*. We use early letters (a, b, c) to denote expressions (constructed from application of the operators) and late letters (x, y, z) to denote variables (within expressions). In an interpretation of a *pKA* expression the variables are mapped to specific (probabilistic) programs. The next definition sets out the details.

$(i)\ 0 + a = a$
$(ii)\ a + b = b + a$
$(iii)\ a + a = a$
$(iv)\ a + (b + c) = (a + b) + c$
$(v)\ a(bc) = (ab)c$
$(vi)\ 0a = a0 = 0$
$(vii)\ 1a = a1 = a$
$(viii)\ ab + ac \leq a(b + c)$ (†)
$(ix)\ (a + b)c = ac + bc$
$(x)\ a \leq b \quad \textit{iff} \quad a + b = b$
$(xi)\ a^* = 1 + aa^*$
$(xii)\ a(b + 1) \leq a \quad \Rightarrow \quad ab^* = a$
$(xiii)\ ab \leq b \quad \Rightarrow \quad a^*b = b$

Programs are denoted by a, b and c. Note that the rule (†) is weaker than the corresponding rule in standard Kleene algebra [7]; this is because of the well-documented [19, 27] interaction of probability and nondeterminism.

Fig. 2. Rules of Probabilistic Kleene algebra, *pKA*.

Definition 3. *The semantic mapping from pKA expressions to $\mathcal{LS}$ is given by*

$$[\![1]\!]_\rho \ \hat{=}\ Id\,, \quad [\![0]\!]_\rho \ \hat{=}\ \top$$
$$[\![ab]\!]_\rho \ \hat{=}\ [\![a]\!]_\rho ; [\![b]\!]_\rho\,, \quad [\![a + b]\!]_\rho \ \hat{=}\ [\![a]\!]_\rho \sqcap [\![b]\!]_\rho\,, \quad [\![a^*]\!]_\rho \ \hat{=}\ [\![a]\!]_\rho^*$$

Here ρ gives the precise interpretation corresponding to the variables in the expressions a and b, so that for variable x, we have $[\![x]\!]_\rho$ is a specific (fixed) probabilistic program $\rho.x$ in $\mathcal{LS}$.

We use $\geq$ for the order in *pKA*, which we identify with $\sqsubseteq_{\mathcal{L}}$ from Def. 2; the next result shows that Def. 3 is a valid interpretation for the rules in Fig. 1, in that theorems in *pKA* apply in general to probabilistic programs.

Theorem 1. *Let ρ be an interpretation as set out at Def. 3. The rules at Fig. 2 are all satisfied, namely if $a \leq b$ is a theorem of pKA set out at Fig. 2, then $[\![b]\!]_\rho \sqsubseteq_{\mathcal{L}} [\![a]\!]_\rho$.*

Proof. Follows from Def. 3, and the fact that $P^* = \lceil \bigcup_{n \geq 0} (Id \sqcap P)^n \rceil$.

Next in Lem. 2 we illustrate some proofs within Kleene algebra of some simple properties of programs. The first two theorems are basic technical equalities; the third equality, on the other hand, is of independent interest as it forms the basis for many "separation"-style theorems common in distributed systems [7], and indeed generalises similar theorems to the probabilistic context.

Lemma 2.

$$a^*a^* = a^* \tag{4}$$
$$a^*(b + c) = a^*(a^*b + a^*c) \tag{5}$$
$$a(b + 1) \leq ca + d \ \Rightarrow\ ab^* \leq c^*(a + db^*) \tag{6}$$

Proof. For (4) we observe from (xi) that $a^* = 1 + aa^*$, thus $aa^* \leq a^*$, and the result follows from $(xiii)$.

For (5) we reason as follows:

$$a^*(a^*b + a^*c) \quad \geq \quad a^*(b+c) \quad = \quad a^*a^*(b+c) \quad \geq \quad a^*(a^*b + a^*c) \ ,$$

where the first inequality follows since $1 \leq a^*$*; the inequality from (4), and the final inequality from* (viii).

Finally for (6) we show first that

$$c^*(a + db^*)(b+1) \leq c^*(a + db^*) \ ,$$

reasoning as follows.

$$\begin{array}{lll}
 & c^*(a + db^*)(b+1) & \\
= & c^*(a(b+1) + db^*(b+1)) & (ix) \\
\leq & c^*(ca + d + db^*(b+1)) & \textit{hypothesis} \\
\leq & c^*(ca + db^* + db^*) & 1, (b+1) \leq b^*\textit{; (4)} \\
= & c^*(ca + db^*) & \textit{(iii)} \\
\leq & c^*(c^*a + c^*db^*) & 1, c \leq c^* \\
= & c^*(a + db^*) \ . & \textit{(5)}
\end{array}$$

From this inequality we now appeal to the induction rule at (xii) to deduce that $c^*(a+db^*)b^* \leq c^*(a+db^*)$*, and the result now follows since* $ab^* \leq c^*(a+db^*)b^*$.

The rules in Fig. 2 purposefully treat probabilistic choice implicitly, and it is only the failure of the equality at (*viii*) which implies that probability may be present in an interpretation $[\![a]\!]_\rho$: in fact it is this property that characterises probabilistic-like models, separating them from those which contain only pure demonic nondeterminism. [6] The use of implicit probabilities fits in well with our applications, where probability is usually confined to statements within a distributed protocol and nondeterminism refers to the arbitrary sequencing of actions that is controlled by a so-called *adversarial scheduler* [27]. For example, if a and b correspond to atomic program fragments (containing probability), then the expression $(a+b)^*$ means that either a or b (possibly containing probability) is executed an arbitrary number of times (according to the scheduler), in any order — in other words it corresponds to the concurrent execution of a and b. Typically a verification of a distributed protocol might involve transformation of a simple, serialised specification architecture, such as a^*b^* (first a executes for an arbitrary number of times, and then b does), into a distributed implementation architecture, such as $(a+b)^*$ using general hypotheses, such as $ab = ba$ (program fragments a and b commute). For instance a typical conjecture might be the following transformation

$$ab \leq ca \quad \Rightarrow \quad ab^* \leq c^*a \ , \tag{7}$$

which says that if a and b are programs such that running a followed by b is a refinement of c followed by a, then it should be the case that a followed

[6] Programming models that include *angelic nondeterminism* as well as demonic nondeterminism satisfy (*viii*), and not the stronger equality [3]; however those models do not satisfy the special limiting properties of probabilistic programs.

by running b for an arbitrary number of times is a refinement of running c similarly for an arbitrary number of times followed by a. Were this result to be proved generally within the proof system then for a particular example where a, b and c were specific programs typically containing precise probabilistic choices, *only the simple hypothesis* $ab \leq ca$ *would need to be checked* (*i.e.* that $[\![ca]\!]_\rho \sqsubseteq_{\mathcal{L}} [\![ab]\!]_\rho$) instead of constructing brute force the concrete model for the whole of the (iterative) programs $[\![ab^*]\!]_\rho$ and $[\![c^*a]\!]_\rho$ and then comparing the results explicitly.

Though plausible, unfortunately the particular conjecture at (7) turns out to be invalid in the probabilistic model (though it is valid in the standard model) and so any attempt to prove otherwise using *pKA* is bound to fail. To see that let ρ be the interpretation such that

$$[\![a]\!]_\rho \quad = \quad [\![b]\!]_\rho \quad = \quad [\![c]\!]_\rho \quad = \quad x{:=}\ 0 \ {}_{\frac{1}{2}}\!\oplus x{:=}\ 1 \tag{8}$$

so that $[\![ab^*]\!]_\rho = x{:=}\ 0 \sqcap x{:=}\ 1$ and $[\![c^*a]\!]_\rho = x{:=}\ 0 \ {}_{\frac{1}{2}}\!\oplus x{:=}\ 1$. Were (7) to be true generally, it would assert (in this case, since $x{:=}\ 0 \sqcap x{:=}\ 1 \sqsubseteq_{\mathcal{L}} x{:=}\ 0 \ {}_{\frac{1}{2}}\!\oplus x{:=}\ 1$ is generally true) the equality of the programs $x{:=}\ 0 \sqcap x{:=}\ 1$ and $x{:=}\ 0 \ {}_{\frac{1}{2}}\!\oplus x{:=}\ 1$. But the result set of the latter program does not contain the point distributions δ_0 and δ_1, whereas the result set of the former does.

Such false conjectures are a common pitfall in the activity of proof, and can be seen as intermediate stages of a validation. Once the error is discovered, the solution is usually clear, and in the case of (7) is fixed by strengthening the hypothesis to $a(b+1) \leq ca$ so that the correct theorem becomes

$$a(b+1) \leq ca \quad \Rightarrow \quad ab^* \leq c^*a\ , \tag{9}$$

which can indeed be verified within the proof system. [7] And the above interpretation at (8) is no longer a counterexample for (9) since the new hypothesis now fails to hold.

Determining which conjectures are false can however be a very time consuming and ad hoc process, thus any automated tool to prompt the user with a counterexample is an invaluable resource. Unfortunately automated counterexample searchers normally use some kind of exhaustive search through models of finite size, but this is not possible for the real-number domain needed to model probability distributions. In the next section we consider an abstraction which, overcomes this problem.

3 Abstract Probabilistic systems

In this section we propose an abstraction of $\mathcal{L}S$ which yields genuinely finite models. The basic idea is to replace a probability distribution d by a simple set, in fact its support $\mathsf{supp.d}$, which contains only the information of which transitions are probabilistic, and the range over which each probabilistic transition extends. We call such a subset the *abstract distribution* associated with d.

[7] Indeed it is a special case of (6) at Lem. 2 above with d set to 0.

This abstraction (mapping distributions to their abstract counterparts) induces an order on subsets of $S^\top$: two subsets (abstract distributions) are defined to be comparable only if there exist corresponding probability distributions which are comparable under $\sqsubseteq_{\mathcal{D}}$. The next definition reformulates that idea without referring to distributions at all.

Definition 4. *Given a distribution d, its associated* abstract distribution *is defined to be* $\mathsf{supp.d}$. *Abstract distributions K and K' are ordered as follows* [8]

$$K \sqsubseteq_{\mathcal{A}} K' \quad \textit{iff} \quad (K = K') \ \vee \ ((\top \in K') \Rightarrow K' \ \subseteq K).$$

The space of abstract programs now uses abstract distributions. The closure conditions are suitable abstractions of those used in Def. 2; in particular union-closure is an abstraction of convex closure.

Definition 5. *The space of* abstract probabilistic programs *is the pair* $(\mathcal{K}S, \sqsubseteq_{\mathcal{K}})$ *where $\mathcal{K}S$ is the set of functions $S^\top \to \wp\wp S^\top$, restricted to subsets which are* union- *and* up *closed with respect to $\sqsubseteq_{\mathcal{A}}$. The order between programs is defined*

$$U \sqsubseteq_{\mathcal{K}} U' \quad \textit{iff} \quad (\forall s\colon S \cdot U.s \supseteq U'.s) \ .$$

Next we define a projection which maps probabilistic programs to abstract probabilistic programs, so that it preserves order.

Definition 6. *The* abstraction projection $\epsilon : \mathcal{L}S \to \mathcal{K}S$ *is defined* $\epsilon.P.s \mathrel{\hat{=}} \{\mathsf{supp.d} \mid \mathsf{d} \in \mathsf{P.s}\}$.

Lemma 3. *For probabilistic programs $P, P'\colon \mathcal{L}S$, if $P \sqsubseteq_{\mathcal{L}} P'$ then $\epsilon.P \sqsubseteq_{\mathcal{K}} \epsilon.P'$.*

Proof. Follows from the definitions of ϵ, $\sqsubseteq_{\mathcal{D}}$ and $\sqsubseteq_{\mathcal{A}}$.

In Fig. 3 we define some mathematical operators over the space of abstract probabilistic programs — they have been chosen so that they correspond via ϵ to the operators for the probabilistic model given at Fig. 1. We say that if K is a subset of abstract distributions then $[\![K]\!]$ is the smallest $\sqsubseteq_{\mathcal{A}}$-up- and union-closed containing K.

By construction the abstraction projection preserves (homomorphically) composition and nondeterminism. In particular it is easy to see that $\epsilon.(P; P') \equiv_{\mathcal{K}} \epsilon.P \mathbin{;} \epsilon.P'$ and that $\epsilon.(P \sqcap P') \equiv_{\mathcal{K}} \epsilon.P [\![] \epsilon.P'$. Our next task is to do the same for iteration — here, as for composition and nondeterminism, our goal is to define the abstract version so that the abstract distributions in the result set of the iteration can be determined by those in the underlying abstract program — even when they correspond to limit distibutions. For example, a probabilistic program modelling of a fair coin, say $\mathsf{coin} \quad \hat{=} \quad \mathsf{head} \ {}_{\frac{1}{2}}\!\oplus \mathsf{tail}$, has the result that its iteration (coin^*) includes both output distributions δ_{head} and δ_{tail} though neither point distribution is a result of any finite number of iterations of coin — it is Cauchy closure that guarantees their inclusion. To see that, we imagine that

[8] This is actually the well-known Hoare order on subsets based on $\sqsubseteq$ for $S^\top$.

$$
\begin{array}{lll}
\textit{Identity} & \mathsf{Id.s} & \hat{=} \llbracket\{\{s\}\}\rrbracket\ , \\
\textit{top} & \top.s & \hat{=} \{\{\top\}\}\ , \\
\textit{composition} & (U ;\!\!\!\!\;\; U').s & \hat{=} \{\bigcup_{u:K} K'_u \mid K{:}\,U.s;\ K'_u{:}\,U'.u\}\ , \\
\textit{probability} & (U \oplus U').s & \hat{=} \llbracket\{K \cup K' \mid K{:}\,U.s; K'{:}\,U'.s\}\rrbracket\ , \\
\textit{nondeterminism} & (U [\!] U').s & \hat{=} \llbracket\{K \mid K{:}\,(U.s \cup U'.s)\}\rrbracket\ ,
\end{array}
$$

Here s is a state in S, and U ,U' are programs, and if K is a subset of abstract distributions then $\llbracket K\rrbracket$ is the smallest $\sqsubseteq_{\mathcal{A}}$-up- and union-closed subset containing K. We deal with iteration below.

Fig. 3. Mathematical operators on abstract probabilistic programs.

the implicit "$\sqcap$" inside of the definition of coin^* acts like a "demon" which can see the value established by the flipped coin after every execution of coin, and can terminate the iteration at any moment — since the laws of probability assert that a fair coin flipped for an arbitrary number of times must *with probability 1* flip a head eventually, the demon can use this to his advantage to wait only long enough until that head appears. The overall effect is that the iteration can terminate with probability 1 in the state head, which is the same as saying that coin^* outputs the point distribution δ_{head}. A similar argument holds for tail.

Thus we need to define the abstract iteration so that $\epsilon.(\mathsf{coin}^*)$ contains the corresponding abstract distributions $\{\mathsf{head}\}$ and $\{\mathsf{tail}\}$.

Definition 7. *For abstract program A in $\mathcal{K}S$, we define A^* as follows. Subset $K \subseteq A^*.s$ if there exists a probabilistic program P in $\mathcal{L}S$ such that $\epsilon.P = A$ and $K = \mathsf{supp.d}$ for some distribution d in $P^*.s$.*

Our next task is to reformulate Def. 7 so that A^* can be determined without referring to any probabilities at all (in an underlying probabilistic program, P say). Fortunately, for finite state spaces S, Lem. 1 implies that the images in ϵ of limit distributions can be characterised in terms of abstract probabilistic properties alone, which is precisely what we need. The next two lemmas set out the details.

Lemma 4. *For any program P in $\mathcal{L}S$, with S finite, we have the equality $\epsilon.(P^*) = (\epsilon.P)^*$.*

Proof. Follows immediately from Lem. 1 which implies that supports of distributions in iterations are independent of the probabilistic weights of the transitions.

Whilst Lem. 4 implies that the actual numeric values of underlying programs P are irrelevant for determining A^* (provided that they are non zero), the next result shows how to compute A^* without referring to the underlying probabilistic programs at all, but only their abstractions. We say that K is *reachable with probability 1 via executions of* A if K is reachable with probability 1 for some program P in $\mathcal{L}S$ such that $\epsilon.P = A$. It is well-known that in finite state spaces there exist algorithms to compute such *probability 1 reachability sets* using only

the information provided by the abstract transitions; for example de Alfaro *et al.* provide such an algorithm [8] with complexity quadratic in the size of the underlying transition system, and we discuss its relevance in the next section.

Lemma 5. *Given abstract program A in $\mathcal{K}S$, where S is a finite state space we can compute A^* as follows. Subset K is in $A^*.s$ if*

1. *$K \subseteq (Id \parallel A)^n$ for some $n \geq 0$;*
2. *or there is some $K' \subseteq A^*.s$ such that $K \subseteq K'$ and for all $k \in K'$ it is possible to reach K with probability 1 from k via executions of $(Id \parallel A)$.*

Proof. Let P be any program in $\mathcal{L}S$ such that $\epsilon.P = A$. The result now follows from Lem. 1.

Finally we can define an interpretation of *pKA* over abstract probabilistic programs so that the two interpretations correspond homomorphically.

Definition 8. *The semantic mapping from pKA terms to the abstract probabilistic semantics is given by* [9]

$$(\!|1|\!)_\rho \mathrel{\hat{=}} \mathsf{Id} \ , \quad (\!|0|\!)_\rho \mathrel{\hat{=}} \top$$
$$(\!|ab|\!)_\rho \mathrel{\hat{=}} (\!|a|\!)_\rho \mathbin{;} (\!|b|\!)_\rho \ , \quad (\!|a+b|\!)_\rho \mathrel{\hat{=}} (\!|a|\!)_\rho \parallel (\!|b|\!)_\rho \ , \quad (\!|a^*|\!)_\rho \mathrel{\hat{=}} (\!|a|\!)_\rho^*$$

Here ρ gives the precise interpretation corresponding to the variables in the expressions a and b, so that for variable x, we have $(\!|x|\!)_\rho$ is a specific (fixed) abstract probabilistic program $\rho.x$ in $\mathcal{K}S$.

The next lemma gives states the relationship between interpretations in $\mathcal{L}S$ and in $\mathcal{K}S$.

Lemma 6. *Let e be any expression in pKA, and let ρ be an interpretation in $\mathcal{L}S$, so that all variables are mapped to programs in $\mathcal{L}S$, and Def. 3 is used to interpret the operators. The equality $\epsilon.[\![e]\!]_\rho = (\!|e|\!)_{\epsilon.\rho}$ holds, where $\epsilon.\rho$ denotes the interpretation in $\mathcal{K}S$ where all variables are mapped to the images under ϵ of the programs defined by ρ, and Def. 8 is used to interpret the operators.*

Proof. Structural induction, Def. 8 and Lem. 5.

In this section we have set up a model for abstract probabilistic programs in which the precise weights attached to the probabilistic transitions have been suppressed, whilst retaining the limiting properties of probability theory.

[9] Note that we do not claim that the Kleene rules are satisfied by this definition; indeed (*xiii*) fails to hold. The abstract program $s_0 \mapsto \{\{s_0, s_1\}\}$; $s_1 \mapsto \{\{s_1\}\}$ denoting both a and b is a counterexample.

4 Towards a framework for counterexample search

In this next section we show how $\mathcal{KS}$ can be used to find counterexamples in $\mathcal{LS}$ using a strategy based on state exploration over finite abstract models.

Lemma 7. *Let e and f be expressions in the Kleene algebra. If $e \neq f$ is satisfiable within $\mathcal{KS}$ then it is also satisfiable within $\mathcal{LS}$.*

Proof. Let the variables in e and f be $x_1, \ldots x_n$. Let ρ be an interpretation which maps each x_i to abstract program A_i within $\mathcal{KS}$ so that $(\!|e|\!)_\rho \neq (\!|f|\!)_\rho$ — this is possible since, by assumption, the inequality is satisfiable in $\mathcal{KS}$. We note that for each A_i there is a corresponding probabilistic model A'_i such that $\epsilon.A'_i = A_i$ (let each abstract distribution K in $A_i.s$ be replaced by the uniform distribution over K). It now follows immediately from Lem. 3 and Lem. 6 that the interpretation defined by the A'_i demonstrates that the inequality is satisfiable in $\mathcal{LS}$ as well.

Finally we have our main result — that if a counterexample exists in $\mathcal{KS}$ to a conjectured equality, it is not provable in pKA.

Corollary 1. *Let e and f be expressions in the Kleene algebra. If $e \neq f$ is satisfiable within $\mathcal{KS}$ then the equality $e = f$ it is not provable by probabilistic Kleene algebra rules.*

Proof. By Lem. 7, the inequality $e \neq f$ is satisfiable within $\mathcal{LS}$, and by Thm. 1 interpretations in $\mathcal{LS}$ satisfy the probabilistic Kleene rules.

To see Lem. 7 in action, consider the assertion at (7). In a two state-space $\{s_0, s_1\}$, we define $A_i.s \mathrel{\hat{=}} \lceil\!\lceil\{\{s_0, s_1\}\}\rceil\!\rceil$ for $i = 1, 2, 3$; the resulting interpretation with a, b, c mapped to A_1, A_2, A_3 respectively show that the negation of (7) is satisfiable in $\mathcal{KS}$. And indeed the construction described in the proof of Lem. 7 shows that it is not satisfiable in $\mathcal{LS}$ either, recovering the counterexample in $\mathcal{LS}$ given at (8).

4.1 Mechanisation of counterexample search

Lem. 7 implies that automated counterexample search for equalities within $\mathcal{LS}$ can be based on state exploration of finite models in $\mathcal{KS}$, and Lem. 5 implies that we can use existing reachability algorithms for finite probabilistic systems to compute (within $\mathcal{KS}$) all instances of a^*.

Based on the model $\mathcal{KS}$ we have implemented a counterexample search using the SAT solving facility [29] of the Isabelle theorem proving environment [25], by translating the pKA expressions into CNF. For very small state spaces, the translation into propositional logic creates an explicit representation of the $*$ operator, where a^* is precomputed for every abstract program a. Due to the number of possible models, this approach is infeasible for larger state spaces, though, fortunately, in practice, counterexamples do appear to be exhibited within very small state spaces. Here however partial evaluation is employed instead to compute a^* symbolically, using an appropriate version of de Alfaro's algorithm applied to

the propositional representation of the abstract program a under consideration. Still the size of the resulting CNF formula limits the size of the models that can be handled. On-the-fly simplification can be used to reduce the size of the formula when a partially known program is considered. Consequently a hybrid approach, where enumeration of models is performed partially by the SAT solver (to reduce the search space) and partially before translation to SAT (to simplify the translation) might prove to be even more efficient.

The fact that the implementation of $*$ is challenging seems to be the case in other systems using SAT solving in the context of $*$-like operators [13].

Finally we note that the proposed procedure for discovering counterexamples outlined above does not, in general, work well for theorems with hypotheses. That is because the abstraction function does not preserve *inequalities*. However, experience with the Kleene algebra has shown that many universal equalities are needed within any proof, thus automated support extending only to equalities is still an important resource.

That said, there are some interesting special cases for which this approach still applies such as hypotheses of the form $p = 0$ [5].

5 Conclusions and comparisons with other approaches

This work represents the first step towards automated reasoning tools for probabilistic distributions systems. Future work will also explore the use of optimisations and heuristics for mechanised search within $\mathcal{KS}$, other strategies to treat hypotheses, and the use of other generalisations of Kleene algebra, to include *termination* properties [7].

Other techniques for verifying probabilistic systems separate probabilistic from standard reasoning [27], but unlike our algebraic approach the standard reasoning only includes properties that are insensitive to the underlying probabilities, and thus only weak properties (typically non probabilistic) can be verified in this way. Other approaches that combine model checking and reasoning to yield parametrised properties include that of Pnueli *et al.* [1]. There is an extensive literature on probabilistic semantics, for example, [18, 27, 15] but as far as we are aware none of this work can support automated counterexample search.

References

1. T. Arons, A. Pnueli, and L. Zuck. Parameterized verification by probabilistic abstraction. In *Proceedings of FOSSACS*, number 2620 in LNCS, April 2003.
2. James Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *J. Algorithms*, 11(3):441–61, 1990.
3. R.-J.R. Back and J. von Wright. *The Refinement Calculus: A Systematic Introduction*. Springer Verlag, 1998.
4. O. Celiku and A. McIver. Cost-based analysis of probabilistic programs mechanised in HOL. *Nordic Journal of Computing*, 2004.

5. E. Cohen. Hypotheses in Kleene Algebra. Bellcore technical report, 1994.
6. E. Cohen. Lazy caching. Bellcore technical report, 1994.
7. E. Cohen. Separation and reduction. In *Mathematics of Program Construction, 5th International Conference*, volume 1837 of *LNCS*, pages 45–59. Springer, 2000.
8. Luca de Alfaro and T. Henzinger. Concurrent ω-regular games. In *Proc. 15th IEEE Symp. Logic in Computer Science*. IEEE, 2000.
9. C. Derman. *Finite State Markov Decision Processes*. Academic Press, 1970.
10. Jifeng He, K. Seidel, and A.K. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28:171–192, 1997.
11. Thai Son Huang. *The Development of a Probabilistic B Method and a Supporting Toolkit*. PhD thesis, Dept. Engineering and Computer Science. In draft.
12. Joe Hurd, A.K. McIver, and C.C. Morgan. Probabilistic guarded commands mechanised in HOL. *Proc. QAPL '04 (ETAPS)*, 2004.
13. D. Jackson. Alloy:A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11:256–290, 2002.
14. C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the IEEE 4th Annual Symposium on Logic in Computer Science*, pages 186–195, Los Alamitos, Calif., 1989. Computer Society Press.
15. B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proc. 6th Conf. LICS*, 1991.
16. D. Kozen. Kleene algebra with tests and commutativity conditions. In *Proceedings of TACAS*, 1996.
17. M. Kwiatkowska, G. Norman, and D.Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *Proceedings of TACAS, 2002*.
18. G. Lowe. Probabilities and priorities in timed CSP. Technical Monograph PRG-111, Oxford University Computing Laboratory, 1993. (DPhil Thesis).
19. A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Programs*. Springer, 2005.
20. C.C. Morgan. The specification statement. *ACM Transactions on Programming Languages and Systems*, 10(3), July 1988. Reprinted in [24].
21. C.C. Morgan. *Programming from Specifications*. Prentice-Hall, 1994.
22. C.C. Morgan. Private communication. 2004.
23. C.C. Morgan, A.K. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, 1996.
24. C.C. Morgan and T.N. Vickers, editors. *On the Refinement Calculus*. FACIT Series in Computer Science. Springer Verlag, Berlin, 1994.
25. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
26. M.O. Rabin. N-process mutual exclusion with bounded waiting by $4 \log 2n$-valued shared variable. *Journal of Computer and System Sciences*, 25(1):66–75, 1982.
27. Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.
28. N. Shankar. Automated verification using deduction, exploration and abstraction. In A.K. McIver and C.C. Morgan, editors, *Programming Methodology*. Springer, 2003.
29. Tjark Weber. Bounded model generation for Isabelle/HOL. In Wolfgang Ahrendt, Peter Baumgartner, Hans de Nivelle, Silvio Ranise, and Cesare Tinelli, editors, *Selected Papers from the Workshops on Disproving and the Second International Workshop on Pragmatics of Decision Procedures (PDPAR 2004)*, volume 125 of *ENTCS*, pages 103–116. Elsevier, July 2005.