Available online at www.sciencedirect.com
ScienceDirect
The Journal of Logic and Algebraic Programming 76 (2008) 90–111
THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING
www.elsevier.com/locate/jlap

Using probabilistic Kleene algebra *pKA* for protocol verification ☆

A.K. McIver [a,*,1], C. Gonzalia [a,1], E. Cohen [b], C.C. Morgan [c,1]

[a] *Department of Computer Science, Macquarie University, NSW 1209, Australia*
[b] *Microsoft Corporation, Redmond, USA*
[c] *Department of Computer Science, University of NSW, NSW 2052, Australia*

Available online 21 December 2007

Abstract

We propose a method for verification of probabilistic distributed systems in which a variation of Kozen's Kleene Algebra with Tests [Dexter Kozen, Kleene algebra with tests, ACM Trans. Programming Lang. Syst. 19(3) (1997) 427–443] is used to take account of the well known interaction of probability and "adversarial" scheduling [Annabelle McIver, Carroll Morgan, Abstraction, Refinement and Proof for Probabilistic Systems, Technical Monographs in Computer Science, Springer-Verlag, New York, 2004].

We describe *pKA*, a probabilistic Kleene-style algebra, based on a widely accepted model of probabilistic/demonic computation [Jifeng He, K. Seidel, A.K. McIver, Probabilistic models for the guarded command language, Sci. Comput. Programming 28 (1997) 171–192; Roberto Segala, Modeling and verification of randomized distributed real-time systems, Ph.D. thesis, MIT, 1995; Roberto Segala, Modeling and Verification of Randomized Distributed Real-Time Systems, PhD thesis, MIT, 1995; Annabelle McIver, Carroll Morgan, Abstraction, Refinement and Proof for Probabilistic Systems, Technical Monographs in Computer Science, Springer-Verlag, New York, 2004]. Our technical aim is to express probabilistic versions of Cohen's separation theorems [E. Cohen, Separation and reduction, in: Mathematics of Program Construction, 5th International Conference, LNCS, vol. 1837, Springer-Verlag, July 2000, pp. 45–59].

Separation theorems simplify reasoning about distributed systems, where with purely algebraic reasoning they can reduce complicated interleaving behaviour to "separated" behaviours each of which can be analysed on its own. Until now that has not been possible for *probabilistic* distributed systems.

We present two case studies. The first treats a simple voting mechanism in the algebraic style, and the second—based on Rabin's *Mutual exclusion with bounded waiting* [Eyal Kushilevitz, M.O. Rabin, Randomized mutual exclusion algorithms revisited, in: Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing, 1992, pp. 275–283]—is one where verification problems have already occurred: the original presentation [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) (1982) 66–75] was later shown to have subtle flaws [I. Saias, Proving probabilistic correctness statements: the case of Rabin's algorithm for mutual exclusion, in: Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing, 1992]. It motivates our interest in algebras, where assumptions relating probability and secrecy are clearly exposed and, in some cases, can be given simple characterisations in spite of their intricacy.

☆ This work extends an earlier report [17].

* Corresponding author.
E-mail address: anabel@ics.mq.edu.au (A.K. McIver).

[1] We acknowledge the support of the Aust. Res. Council Grant DP0558212.

1567-8326/$ - see front matter
doi:10.1016/j.jlap.2007.10.005

Finally we show how the algebraic proofs for these theorems can be automated using a modification of Aboul-Hosn and Kozen's KAT-ML [Kamal Aboul-Hosn, Dexter Kozen, KAT-ML: An interactive theorem prover for Kleene algebra with tests, J. Appl. Non-Classical Logics 1 (2006)].

Keywords: Kleene algebra; Probabilistic systems; Probabilistic verification

1. Introduction

The verification of probabilistic systems creates significant challenges for formal proof techniques. The challenge is particularly severe in the distributed context where quantitative system-wide effects must be assembled from a collection of disparate localised behaviours. Here carefully prepared probabilities may become inadvertently skewed by the interaction of so-called adversarial scheduling, the well-known abstraction of unpredictable execution order of the distributed components' behaviours. Indeed whilst this interaction and possible skewing effects can lead to violation of the protocol's specification [26], the source of the error may be very difficult to locate.

One approach to the verification problem is probabilistic model checking [25], but it often quickly becomes overwhelmed by state-space explosion, and so verification is often possible only for small problem instances. On the other hand quantitative proof-based approaches [20,11], though in principle independent of state-space issues, may similarly fail due to the difficulties of calculating complicated probabilities, effectively "by hand".

In this paper we propose a third way, in which we combine some proof with numerical computations. The idea is to apply algebraic proofs in a "pre-processing" stage to simplify a distributed architecture *without the need to do any numerical calculations whatsoever*. When numerical calculations are required, they can then be applied to the significantly simpler "serialised systems", where more general analytic results can often be found [12]. The proposed method is based on *reduction*, the well-known strategy of simplifying distributed algorithms, *but applied in the probabilistic context*. In this context, the demonstration that a complex distributed scenario is equivalent to a simpler serial one is effectively the same as showing that the adversarial scheduler is unable to skew the probabilities in an undesirable manner.

Weak Kleene algebra, pKA

We review the program algebra *pKA* [18] in which standard *Kleene algebra* [14] has been adapted to reflect the interaction of probabilistic assignments with nondeterminism, where the latter is used to model the unpredictable actions of the scheduler. Standard (i.e. non-probabilistic) Kleene algebra his been used effectively to verify some non-trivial distributed protocols [6], and we will argue that the benefits carry over to the probabilistic setting as well. The main difference between *pKA* and standard Kleene algebra is that *pKA* disallows certain distributive laws, just in those cases where the interaction of the probability and nondeterminism is an issue [29,9,20]. The impoverished algebra is then repaired by replacing those axioms with adjusted (weaker) probabilistic versions.

Algebraic specification and verification

Our main case study is inspired by Rabin's solution to the mutual exclusion problem with bounded waiting [26,15], whose original formulation was found to contain some subtle flaws [28] due precisely to the combination of adversarial and probabilistic choice we address. Later it became clear that the assumptions required for the correctness of Rabin's probabilistic protocol—that the outcome of some probabilistic choices should be invisible to the adversary—cannot be supported by the Markov Decision Process-style model, now generally accepted for probabilistic systems [9,20,25]. We discuss the implications on the model and algebra of adopting those assumptions which, we argue, have wider applications for secrecy and probabilities.

Computer support for pKA

One advantage of the algebraic approach is the opportunity for computer support for the proofs. Not only can a proof script act as a "certificate", increasing the confidence in the result, but a sophisticated interactive tool can automate many of the steps required in a fully formal proof. In the context of probabilistic systems, mechanised proof is a rarity, with the combination of real-number arithmetic and complex Boolean expressions militating against extensive automation—whilst sequential programs can be mechanised and verified in detail [5], more complex distributed protocols are significantly more challenging. In terms of practical verification, using a qualitative algebra to simplify the form of a protocol to one which is essentially sequential—*a task which in itself can easily be automated*—brings the verification of probabilistic distributed protocols within range of full automation after all.

In this paper we consider the automation of the simplification stage, and base it on KAT-ML, a tool originally designed by Aboul-Hosn and Kozen [3] for reasoning within standard Kleene algebra with tests. We show that a modification of that tool can be used to provide such support even for *pKA*—we describe the modification and use it to automate all the proofs of the main algebraic theorems used for our protocol verification.

Overall, our specific contributions in this paper are as follows:

1. A summary of *pKA*'s characteristics (Section 2), including a generalisation of Cohen's work on separation [6] for probabilistic distributed systems using *pKA* (Section 4);
2. A "tutorial-style" description of how the algebraic approach may be applied in probabilistic verification of a simple voting scheme (Section 3);
3. Application of the general separation results to Rabin's solution to distributed mutual exclusion with bounded waiting (Section 7);
4. A description of how the algebraic proofs may be automated in an adapted version of Aboul-Hosn and Kozen's KAT-ML system [3] (Section 5).

We end in Section 8 with a discussion of how the algebraic approach may be used for simple specifications of the critical "secrecy" properties mentioned above.

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]. A point distribution centered at a point k is denoted by δ_k, and is defined so that $\delta_k.k' \mathrel{\hat{=}} 1$ just when $k = k'$. The $(p, 1-p)$ -weighted average of distributions d and d' is denoted $d\ {}_p\!\oplus d'$, and is defined to be the distribution given by $(d\ {}_p\!\oplus d').k \mathrel{\hat{=}} p\times d.k + (1-p)\times d'.k$. If K' is a subset of states, and d a distribution, we write $d.K'$ for $\sum_{k\in K'} d.k$. The power set of K is denoted $\mathbb{P}K$. We use early letters a, b, c for general Kleene expressions, late letters x, y for variables, and t for tests (introduced below).

2. Probabilistic Kleene algebra

Given a (discrete) state space S, the set of functions $S \to \mathbb{P}\overline{S}$, from (initial) states to subsets of distributions over (final) states has now been thoroughly worked out as a basis for the transition-system style model now generally accepted for probabilistic systems [20] though, depending on the particular application, the conditions imposed on the subsets of (final) probability distributions can vary [24,9]. 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 interleaved events in a distributed system). The functions $S \to \mathbb{P}\overline{S}$ model the unquantifiable aspects with powersets ($\mathbb{P}(\cdot)$) and the quantifiable aspects with distributions ($\overline{S}$).

For example, a program that simulates a fair coin is modelled by a function that maps an arbitrary state s to (the singleton set containing only) the distribution weighted evenly between states 0 and 1; we write it

$$\mathit{flip} \quad \mathrel{\hat{=}} \quad s := 0\ {}_{1/2}\!\oplus\ s := 1, \tag{1}$$

which we can abbreviate $s := 0\ {}_{1/2}\!\oplus 1$.

In contrast a program that simulates a possible 0-bias of at most $1/6$ is modelled by a nondeterministic choice delimiting a range of behaviours:

$$\mathit{biasFlip} \;\hat{=}\; s := 0 \,{}_{1/2}\!\oplus 1 \;\sqcap\; s := 0 \,{}_{2/3}\!\oplus 1, \tag{2}$$

and in the semantics (given below) its result set is represented by the *set* of distributions defined by the two extremal probabilities $1/2$ and $2/3$.

In setting out the details, we follow Morgan et al. [24] and take a domain theoretical approach, restricting the result sets of the semantic functions according to an underlying order on the state space. We take a "(co-) flat" domain $(S^\top, \sqsubseteq)$, where $S^\top$ is $S \cup \{\top\}$ (in which $\top$ is a special state used to model miraculous behaviour) and the order $\sqsubseteq$ is constructed so that $\top$ dominates all (proper) states in S, which are otherwise unrelated.

The principal reason for using this domain is that we wish only to model "normal" program operation (i.e. the result of state changes) together with miracles, and both those types of behaviours are captured adequately by our semantic model. We note that our use of the special $\top$ state becomes necessary only in the probabilistic context—standard relational models of programming do not require it, using instead the "empty result set" to indicate miraculous behaviour. In generalising those ideas to include probability the problem is that we cannot formalise distributions over empty results; but our introduction of an explicit $\top$ suffices [22].

Definition 1. Our probabilistic power domain is a pair $(\overline{S^\top}, \sqsubseteq)$, where $\overline{S^\top}$ is the set of normalised functions from $S^\top$ into the real interval $[0, 1]$, and $\sqsubseteq$ is induced from the underlying "co-flat" order $\sqsubseteq$ on $S^\top$ so that

$$d \sqsubseteq d' \quad \text{iff} \quad (\forall K \subseteq S \cdot d).K^\top \;\leq\; d'.K^\top.^{2}$$

Probabilistic programs are now modelled as the set of functions from initial state in $S^\top$ to sets of final distributions over $S^\top$, where the result sets are restricted by so-called *healthiness conditions* characterising viable probabilistic behaviour, motivated in detail elsewhere [20]. They are *up-closure*, *convex-closure* and *Cauchy closure*. A set D of distributions is *up-closed* if whenever $d \in D$, and $d \sqsubseteq d'$, then $d' \in D$; a set D of distributions is *convex-closed* if whenever $d, d' \in D$, then their p-weighted average $d \,{}_p\!\oplus d' \in D$, for any real $0 \leq p \leq 1$; a set D of distributions is *Cauchy closed* if it contains all its limit points.[3] These healthiness conditions mean that the semantics accounts for specific features of probabilistic programs by construction. For example 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"[4]) is also accounted for (implied by Cauchy closure). A further bonus is that, as for standard relational models of programs, program refinement is simply defined as reverse set-inclusion. We observe that probabilistic properties are preserved with increase in this order.

Definition 2. The space of probabilistic programs is given by $(\mathcal{L}S, \sqsubseteq)$ where $\mathcal{L}S$ comprises those $\top$-preserving 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 $\sqsubseteq$, where $\top$-preserving maps $\top$ to $\{\delta_\top\}$. The order between programs is defined

$$\mathit{Prog} \sqsubseteq \mathit{Prog}' \quad \text{iff} \quad (\forall s \in S \cdot \mathit{Prog}.s \supseteq \mathit{Prog}'.s),$$

where both $\mathit{Prog}, \mathit{Prog}' \in \mathcal{L}S$, and thus are functions $S^\top \to \mathbb{P}S^\top$ as above.

For example the healthiness conditions mean that the semantics of the program at (2) contains all mappings of the form

$$s \mapsto \delta_0 \,{}_q\!\oplus \delta_1, \quad \text{for } 2/3 \geq q \geq 1/2,$$

where respectively δ_0 and δ_1 are the point distributions on the states $s = 0$ and $s = 1$.

[2] The definition of $\sqsubseteq$ is equivalently $(\forall s{:}\, S \cdot d.s \geq d.s')$ because distributions are 1-summing. We leave it as it is because it shows more clearly the connection with the probabilistic powerdomain construction [13].

[3] In a metric space a limit point d of a set D is a point such that, given any $\epsilon > 0$, there is some $d_\epsilon \in D$ with $\mathsf{dist}(d, d_\epsilon) < \epsilon$, where dist is the distance function defining the metric. In our space of distributions, where the state space is finite we can define that distance as the maximum of $|d.k - d'.k|$, taken over all states k (including $\top$).

[4] 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 '*flip*' inside an iteration, which is discussed below.

Skip	$\mathsf{skip}.s$	$\hat{=}$	$\lceil\{\delta_s\}\rceil$
Miracle	$\mathsf{magic}.s$	$\hat{=}$	$\{\delta_\top\}$
Chaos	$\mathsf{chaos}_K.s$	$\hat{=}$	$\overline{K^\top}$
Composition	$(Prog; Prog').s$	$\hat{=}$	
			$\{\sum_{u\in S^\top} d.u \times D'.u \mid d \in Prog.s; \forall v \in S \cdot D'.v \in Prog'.v\}$
Choice	$(\mathsf{if}\ B\ \mathsf{then}\ Prog\ \mathsf{else}\ Prog').s$	$\hat{=}$	
	$\text{if } B.s \text{ then } Prog.s \text{ else } Prog'.s$		
Probability	$(Prog\ {}_p\!\oplus\ Prog').s$	$\hat{=}$	$\{d\ {}_p\!\oplus\ d' \mid d \in Prog.s; d' \in Prog'.s\}$
Nondeterminism	$(Prog\ \sqcap\ Prog').s$	$\hat{=}$	$\lceil Prog.s \cup Prog'.s\rceil$,
Iteration	$Prog^*$	$\hat{=}$	$(\nu X \cdot Prog; X\ \sqcap\ \mathsf{skip})$

In the above definitions s is a state in S, $K \subseteq S$, and B is a Boolean-valued function of the state. For any set $D \subseteq \overline{S^\top}$, the expression $\lceil D\rceil$ denotes the smallest up-, convex- and Cauchy-closed subset of distributions containing D. Programs are denoted by *Prog* and *Prog*′, and the expression $(\nu X \cdot exp)$ denotes the greatest fixed point of the function $(\lambda X \cdot exp)$, in the case above a program-to-program function. Note that all programs map $\top$ to $\{\delta_\top\}$.

Fig. 1. Mathematical operators on the space of programs [20].

In Fig. 1 we define some mathematical operators on the space of programs: they will be used to interpret our language of Kleene terms. Informally composition *Prog* ; *Prog*′ corresponds to a program *Prog* being executed followed by *Prog*′, so that from initial state s, any result distribution d of *Prog*.s can be followed by an arbitrary distribution of *Prog*′. The probabilistic operator ${}_p\!\oplus$ takes the average of the distributions of each of its operands, weighted according to the real-value p; the nondeterminism operator takes the union of its operands' results (with closure).

Iteration is the most intricate of the operations—operationally *Prog** represents the program that can execute *Prog* an arbitrary number of times, but must stop iterating at some point; however the "decision" whether or not to continue iterating is effectively made "on-the-fly". In the probabilistic context, as well as generating the results of all "finite iterations" of (*Prog* $\sqcap$ skip) (*viz.* a finite number of compositions of (*Prog* $\sqcap$ skip)), imposition of Cauchy closure acts as usual on metric spaces in that it also generates all *limiting* distributions—i.e. if $d_0,\ d_1, \ldots$ are distributions contained in a result set U, and they converge to d, then d is contained in U as well. The intuition for including limit distributions in the semantics is based on the observation that there is no test which can effectively distinguish between a program which can establish a specific condition "with probability 1" and one which can establish the same condition "with probability arbitrarily close to 1"—thus (by imposing Cauchy closure) we ensure that the semantics does not distinguish those cases either.

To illustrate, we consider

$$\mathit{halfFlip} \ \hat{=}\ \mathsf{if}\ \ s{=}0\ \ \mathsf{then}\ \ \mathit{flip}\ \ \mathsf{else}\ \ \mathsf{skip}, \tag{3}$$

where *flip* was defined at (1). It is easy to see that the iteration *halfFlip** corresponds to a transition system which can (but does not have to) flip the state from $s = 0$ an arbitrary number of times. Thus after N iterations of *halfFlip*, the result set contains all the distributions $\delta_0\ {}_{1/2^n}\!\oplus\ \delta_1$ for $n \leq N$. Cauchy Closure then implies the result distribution set must contain δ_1 as well.

We shall repeatedly make use of *tests*, defined as follows. Given B, a Boolean-valued function of the state space, we write $[B]$ for the test

$$\mathsf{if}\ \ B\ \ \mathsf{then}\ \ \mathsf{skip}\ \ \mathsf{else}\ \ \mathsf{magic}, \tag{4}$$

viz. the program which skips if the initial state satisfies B, and behaves like a miracle otherwise. We use $[\neg B]$ for the *complement* of $[B]$. Tests are standard (non-probabilistic) programs which, given the definitions at Fig. 1 satisfy the following properties:

- $[B]; [\neg B] = \mathsf{magic}$—a test followed by its negation results in a miracle (and is a "zero" of Composition);
- $[B] \sqcap [\neg B] = \mathsf{skip}$—the nondeterministic choice between a test and its negation is the "identity" of Composition;
- $\mathsf{skip} \sqsubseteq [B]$—the identity is refined by any test.
- *Prog* ; $[B]$ determines the probabilities with which *Prog* may establish B. To see that, let d be any distribution in the result set of $(\mathit{Prog}\,;\ [B]).s_0$, where s_0 is some initial state. The presence of the test $[B]$ means that for any (proper) state $s \neq \top$, we can have $d.s > 0$ only if s "was allowed through" by the skip part of $[B]$, i.e. satisfies B. Hence the sum $d.S$ gives the *overall probability* that the final result of executing *Prog* from s_0 satisfies B in that case.
  For example consider *biasFlip* at (2). The program *biasFlip* ; $[s = 0]$ is

  $$(s := 0)\ {}_{1/2}\!\oplus \mathsf{magic} \sqcap (s := 0)\ {}_{2/3}\!\oplus \mathsf{magic} \quad = \quad (s := 0)\ {}_{2/3}\!\oplus \mathsf{magic},$$

  a program whose probability $(2/3)$ of not being blocked (by a miracle) is the *maximum* probability with which *biasFlip* can establish $s = 0$, since the demonic choice is resolved to avoid (miraculous) blocking.
- Similarly, *Prog* ; $[B]$; $\mathsf{chaos}_K = \mathsf{chaos}_K\ {}_{p_s}\!\oplus \mathsf{magic}$, where p_s is the greatest probability that *Prog* may establish B from initial state s, because chaos_K masks all information except for the probability that the test is successful. For example in the example above, we have *biasFlip* ; $[s = 0]$; chaos_S is

  $$(s := 0\ {}_{2/3}\!\oplus \mathsf{magic});\ \mathsf{chaos}_S \quad = \quad \mathsf{chaos}_S\ {}_{2/3}\!\oplus \mathsf{magic}.$$

- Tests distribute over nondeterministic choice since, more generally, if *Prog* contains no probabilistic choice, then *Prog* distributes over $\sqcap$, i.e. for any Prog' and Prog'', we have

  $$\mathit{Prog}\ ;\ (\mathit{Prog}' \sqcap \mathit{Prog}'') = \mathit{Prog}\ ;\ \mathit{Prog}' \sqcap \mathit{Prog}\ ;\ \mathit{Prog}''.$$

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

2.1. Mapping pKA into $\mathcal{LS}$

In our use of Kleene algebra the variables denote programs, with distinguished programs 1 and 0; and the operators comprise sequential composition (having identity 1 and zero 0), a binary plus $+$ and unary star $*$. Terms are ordered by $\leq$ induced from $+$ (see Fig. 2), and both binary as well as the unary operators are monotone with respect to that order. 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*. These are the same as standard Kleene algebra, except for the (weaker) rules indicated by (†) and (‡); we justify the weakening below. We shall also use *tests*, whose denotations are programs of the kind (4). We normally denote a test by t, and its complement is $\neg t$.

The next definition gives an interpretation of *pKA* in $\mathcal{LS}$.

Definition 3. Assume that for all variables x the denotation $[\![x]\!] \in \mathcal{LS}$ as a program (including tests) is given explicitly. We interpret the Kleene operators over terms as follows:

$$[\![1]\!] \mathrel{\hat{=}} \mathsf{skip}, \quad [\![0]\!] \mathrel{\hat{=}} \mathsf{magic},$$
$$[\![ab]\!] \mathrel{\hat{=}} [\![a]\!]; [\![b]\!], \quad [\![a + b]\!] \mathrel{\hat{=}} [\![a]\!] \sqcap [\![b]\!], \quad [\![a^*]\!] \mathrel{\hat{=}} [\![a]\!]^*.$$

Here a and b stand for other terms, including simple variables.

The order $\leq$ of *pKA* is identified with $\sqsupseteq$ from Definition 2 (note reversal); for example $0 \leq 1$ corresponds to $[\![0]\!] \sqsupseteq [\![1]\!]$, that is $\mathsf{magic} \sqsupseteq \mathsf{skip}$.

The next result shows that Definition 3 is a valid interpretation for the rules in Fig. 1, in that theorems in *pKA* apply in general to probabilistic programs.

Theorem 4 [18]. *Let* $[\![\cdot]\!]$ *be an interpretation as set out at Definition* 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 indeed* $[\![b]\!] \sqsubseteq [\![a]\!]$.

To see why we cannot have equality at (†) in Fig. 2, consider the expressions $a(b+c)$ and $ab+ac$ with $[\![a]\!] = \mathit{flip}$ and $[\![b]\!] = \mathsf{skip}$ and $[\![c]\!] = (s := 1{-}s)$. Then $[\![a(b+c)]\!]$ is

$$s := 0 \,{}_{1/2}\!\oplus 1;\ (\mathsf{skip} \sqcap s := 1{-}s),$$

where we can see clearly that the demonic choice is made after the probabilistic choice in *flip* has been resolved. This means that one possible result is for s to be assigned 0 with probability 1, since the following valid refinement

$$s := 0 \,{}_{1/2}\!\oplus 1; \quad \mathsf{if}\ \ s{=}0\ \ \mathsf{then}\ \ \mathsf{skip}\ \ \mathsf{else}\ \ s := 1{-}s,$$

is semantically equivalent to $s := 0$ (each probabilistic branch is followed by a program which has the effect of setting s to 0).

On the other hand, denotation $[\![ab+ac]\!]$ is the program

$$\begin{array}{l} s := 0 \,{}_{1/2}\!\oplus 1;\ \mathsf{skip} \ \sqcap\ s := 0 \,{}_{1/2}\!\oplus 1;\ s := 1{-}s \\ = s := 0 \,{}_{1/2}\!\oplus 1 \ \sqcap\ s := 0 \,{}_{1/2}\!\oplus 1 \\ = s := 0 \,{}_{1/2}\!\oplus 1, \end{array}$$

which clearly does not have the property that s may be set to 0 with probability 1. In the second case the demonic choice is resolved *before* the probabilistic choice, and thus cannot exploit the result of the subsequent random flip.

In summary, the failure of distribution (of a) over the demonic choice indicates that there is more information available to the demon after execution of a than before.

Similarly the rule at Fig. 2 (‡) is not the usual one for Kleene algebra. Normally this induction rule only requires a weaker hypothesis, but that (weaker) rule, $ab \le a \Rightarrow ab^* = a$, is unsound for the interpretation in $\mathcal{LS}$ for similar reasons.

Consider the interpretation where each of a, b and c represent *flip* defined at (1). We show below that $\mathit{flip}^* = s := 0 \sqcap 1$ (using a similar abbreviation), so that

$$\mathit{flip}\ ;\ \mathit{flip}^* \quad = \quad s := 0 \sqcap 1 \quad \neq \quad \mathit{flip},$$

in spite of the fact that $\mathit{flip}\ ;\ \mathit{flip} = \mathit{flip}$.

To see why flip^* results in the nondeterministic choice,
we reason

$$\begin{array}{ll} \mathit{flip}^* & \\ = (\nu X \cdot \mathit{flip}; X \ \sqcap\ \mathsf{skip}) & \\ \sqsubseteq (\nu X \cdot \mathsf{if}\ \ s{=}1\ \mathsf{then}\ \ \mathit{flip}; X\ \mathsf{else}\ \ \mathsf{skip}) & \\ = s := 0, & \text{because Cauchy closure includes the limit point} \end{array}$$

and note that similar reasoning yields $\mathit{flip}^* \sqsubseteq s := 1$.

The first refinement represents the case where the demon continues to execute *flip* until the result $s = 0$ is achieved, and in the second the demon strives for 1. Thus we establish $\mathit{flip}^* \sqsubseteq s := 0 \sqcap 1$ at least (sufficient to make our point), though in fact they are equal.

For the sound rule (‡), the antecedent fails. Indeed the effect of the $(1+b)$ in that rule is to capture explicitly the action of the demon, and the hypothesis is satisfied only if the demon cannot skew the probabilistic results in the way illustrated above.

Whilst Theorem 4 establishes the soundness of the rules at Fig. 2 for probabilistic semantics, we note that there are other nonprobabilistic models which also satisfy them. These include *Monodic tree languages* [30] (in fact having the same equational theory as *pKA*); programming models accommodating both angelic and demonic choices [4]; and other formulations of programming frameworks such as the Lazy Kleene Algebras (which among other things provides an abstraction of predicate transformers) [21]. However in the current context—restricted only to nondeterminism and probability (although the latter only implicitly)—*pKA*'s lack of distributivity through nondeterministic choice indicates that there is a conflict between the two types of branching construct, and thus distinguishes between the presence, or absence of probability.

The use of implicit probabilities fits in well with our applications, where probability is usually confined to code residing at individual processors within a distributed protocol and nondeterminism refers to the arbitrary sequencing of actions that is controlled by a so-called *adversarial scheduler* [29]. For example, if a and b correspond to atomic

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

Fig. 2. Rules of Probabilistic Kleene algebra *pKA* [18].

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), and in any order. In other words it corresponds to the concurrent execution of a and b.

Typically a two-stage verification of a probabilistic distributed protocol might involve first the transformation of a distributed implementation architecture, such as $(a+b)^*$, to a simple, separated specification architecture, such as a^*b^* (first a executes for an arbitrary number of times, and then b does), using general hypotheses, such as $ab \leq ba$ (program fragments a and b sub-commute). The second stage would then involve a model-based analysis in which the hypotheses postulated to make the separation go through would be individually validated by examining the semantics in $\mathcal{LS}$ of the precise code for each.

In the next section we illustrate how the combination of the algebraic and semantic approaches overall simplifies the verification of a straightforward probabilistic voting mechanism. In Section 6 we describe how the method can be used for a much more intricate protocol.

3. A simple voting scheme

Probability can offer an attractive implementation mechanism for distributed voting; and our algebraic approach can be used to simplify the verification task: that despite adversarial scheduling, the voting protocol is still impartial. We illustrate this with a simple example.

> N processes wish to decide which one of them is to be granted exclusive access to a critical section. They do so by executing a distributed election protocol, where an adversarial scheduler "moderates" which of the processes is allowed to participate. The protocol must be designed to meet the following fairness criterion: *any pair of processes have equal chance of winning any competition in which they both take part*.

It is normal practice to assume that the scheduler is not only able to choose the execution order of the processes, but also *which ones* to schedule in any competition—the only expected commitment is that eventually each process must be scheduled [27]. Under that weak constraint, the above additional fairness condition ensures that the scheduler cannot favour one process over another.

A simple scheme to implement distributed voting is set out at Fig. 3. Each process, if selected, first checks whether it has already voted, and if it has not, sets itself to be the winner, via the assignment "$w := x$" with probability $1/n$, after incrementing the variable n recording the current number of participants. The behaviour of the whole system—comprising a (finite) set $\mathcal{X}$ of processes—is specified using the Kleene star over their generalised sum[5],

$$\textit{Election} \quad \hat{=} \quad (+_{x\in\mathcal{X}} V_x)^*, \tag{5}$$

where the adversarial choice is modelled by the nondeterministic choice. In program execution terms, we are saying that each process may execute its voting program "for a while".[6]

[5] We use $+_{i\in\mathcal{I}} a_i$ for the generalised nondeterministic choice over programs a_i, where i is drawn from a finite index set $\mathcal{I}$.

[6] Here we are making the (possibly too strong) assumption that the code in Fig. 3 is executed "atomically".

V_x : if $\neg v_x$ then	*If x has not yet participated ...*
$v_x := true;$	*record his participation ...*
$n := n+1;$	*increase the participation count ...*
$w := x \ {}_{1/n}\!\oplus \mathsf{skip}$	*and now flip to win.*

The variables w and n record respectively the current winner, and the number of participants; v_x is a local Boolean registering x's having participated in the current vote.

Fig. 3. Local code for node x in a distributed voting scheme.

The intuition behind this straightforward scheme is based on a simple property of probability as follows. Observe first that the later a process casts its vote, the lower is the chance of winning, but that the probabilities are carefully chosen so that the overall fairness is unaffected by the specific voter line-up. The reason is that the apparent advantage for early voters is offset by the (repeated) chance that the current winner is usurped.

For example, we can prove by an inductive argument that after m participants $V_0, \ldots V_{m-1}$ have voted independently, then the probability that $w = i$, for $0 \leq i < m$ is $1/m$. The base case is dealt with trivially, for m is assumed to be 0 initially, and then w is set to the identity of the first participant (say V_0) after first incrementing n. The overall effect of that is to assign w the value 0 with probability $1 = 1/1$, and n the value of 1.

For the inductive step, we assume that m participants have voted, so that n is m, and that the result is the uniform probabilistic assignment to w over m values, which we denote by the *generalised probabilistic choice* as follows; the program so far is effectively

$$n := m; \ \bigoplus_{0\leq i<n} w := i \ @1/n,$$

where we have introduced a notation for generalised probabilistic choice. If participant V_m is now selected, the program so far becomes

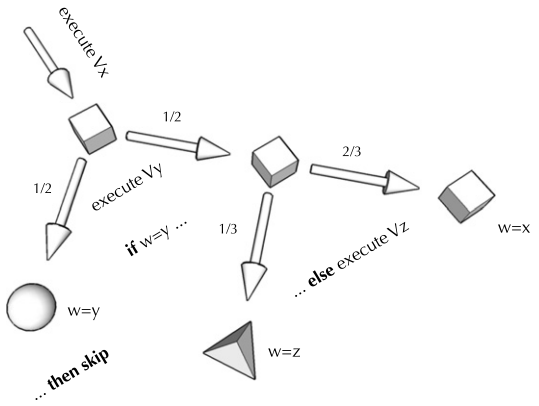
$$\begin{aligned}
&n := m;\ (\textstyle\bigoplus_{0\leq i<n} w := i \ @1/n);\ n := n{+}1;\ (w := n \ {}_{1/n}\!\oplus \mathsf{skip})\\
=\ &n := m{+}1;\ (\textstyle\bigoplus_{0\leq i<n-1} w := i \ @1/(n{-}1));\ (w := n \ {}_{1/n}\!\oplus \mathsf{skip})\\
=\ &n := m{+}1;\ (\textstyle\bigoplus_{0\leq i<n-1} w := n \ {}_{1/n}\!\oplus i \ @1/(n{-}1))\\
=\ &n := m{+}1;\ w := n \ {}_{1/n}\!\oplus (\textstyle\bigoplus_{0\leq i<n-1} w := i \ @1/(n{-}1))\\
=\ &n := m{+}1;\ (\textstyle\bigoplus_{0\leq i<n} w := i \ @1/n),
\end{aligned}$$

thus establishing the induction. Finally we note that this general property ensures the pairwise fairness criterion stated above.

Unfortunately the above argument does not take adversarial scheduling into account, in particular (5) that the adversary not only controls the *order*, but also the *list* of participants. As explained above such freedom is typically considered in the system model during verification so that the verification covers "worst case" scenarios—protocols which can withstand this level of scheduler interference would be deemed to be very robust indeed. In this case the analysis (set out below) shows that the protocol at Fig. 3 is not resilient against such disturbance and indeed the adversary can favour one process over another by exercising its control. This indicates that either the adversary's freedom must be curtailed, or the protocol significantly strengthened.

To see an example of what can happen, consider the run of *Election* set out at Fig. 4 in a system comprising 3 voters. Here the adversary in $(V_x + V_y + V_z)^*$ executes V_x, then V_y; but he goes on to execute V_z only if x is still the current winner. The probabilistic behaviour of this schedule can be seen from Fig. 4—and a brief examination shows that it presents a counterexample to the pairwise fairness criterion. For example, suppose we select competitions in which both y and z participate—in Fig. 4 we see that only happens in the case that y has already lost, thus the chance that y wins (restricted to those competitions) is 0, whereas z's chance of winning is $1/3$.[7]

[7] This is using the standard probabilistic technique of conditioning applied to the tree in Fig. 4: the probability that y wins *given* that both y and z participate is 0.



The adversary may choose to schedule x, y or z in any order (if at all). In this example, he chooses to schedule x, then y — and then possibly z, but *only* in the case that y has lost:

□: x is winning;
◯: y is winning;
△: z is winning.

Fig. 4. A voting run with adversarial choice.

The underlying problem here is that with such powerful adversarial scheduling, the fact that the scheduler has knowledge of the result of previous probabilistic choices matters—a so-called oblivious scheduler (in which the scheduler's behaviour cannot be correlated with the results of probabilistic branching) would not gain that advantage, and similarly would not be able to discriminate against y in the way illustrated here.[8] The fact that we do not have oblivious scheduling here in Fig. 4 is a depiction of the failure of distributivity

$$V_x V_y(1 + V_z) \quad \neq \quad V_x V_y + V_x V_y V_z,$$

for the right-hand side does not include the possibility of V_z being scheduled on the basis of V_y's luck, as the left-hand side does. Were the distributivity to hold, we would easily be able to show that *Election* is equivalent to distributing all the nondeterminism $(+)$ to the front of the expression, so that the scheduler would be forced to decide beforehand which processes to schedule for the competition. Indeed, that being so, we would have proved the pairwise fairness criterion after all. In *pKA* however that possibility does not exist precisely because in probabilistic programming schedulers are not in general oblivious, and can only made to appear so by careful protocol design.

This example illustrates the difficulties lurking in a probabilistic analysis (and indeed in specifications[9]), even within apparently very simple protocols. In this case the problem is that the scheduler is too powerful, able to construct his schedule on the basis of the outcome of the coin flips. The intention however, is to force the scheduler to behave obliviously, namely that his schedule should be somehow chosen independently of probabilistic outcomes.

One simple way to avoid the scheduler deciding to exclude a process is to enforce a condition that all three must participate. We do this with the test $T \mathrel{\hat{=}} v_x \wedge v_y \wedge v_z$, and then define the distributed protocol to be

$$\textit{3-Election} \quad \hat{=} \quad (V_x + V_y + V_z)^*\,[T], \tag{6}$$

which now excludes the schedule depicted in Fig. 4, where in one of the possible executions, V_z is scheduled, and in another it is not.

We can explain (6) in terms of the semantics as follows. First, the Kleene star operator accounts for all *possible* actions of the scheduler, but then the following test "winnows out" all schedules that fail T. In the case that any of the three v_i's is *false* initially, the test $[T]$ thus enforces a final result where $v_i = \textit{true}$, indicating that participant V_i was indeed scheduled.[10]

Next we make the allusion to fairness precise by decreeing an election to be fair if it yields the same result as the serialisation $V_x V_y V_z$—which as we have seen sets each process to be winner with probability $1/3$. Thus *3-Election* is fair if the refinement

[8] It turns out however that "oblivious scheduling" does have a neat algebraic characterisation as a distribution law involving $+$, and we discuss this later in Section 8.

[9] The problems with such informal specifications, especially in the probabilistic context, have been pointed out by Saias [28].

[10] In this example the scheduler may schedule each participant V_i with probability 1, and if he does so, inspection of the code at Fig. 3, implies that the corresponding guard v_i must be set to *true* with probability 1 as well.

For all permutations i, j, k of x, y, z we have

(a) Candidates may vote only once: $V_i(1 + V_i) = V_i$.

(b) Voting is independent: $V_i V_j = V_j V_i$.

(c) The final tally is independent of voter order:

$$(1+V_x)(1+V_y)(1+V_x)[T] \quad \leq \quad (1+V_i)(1+V_j)(1+V_k)[T] \quad \leq \quad V_i V_j V_k \ ,$$

Here T is the predicate $v_x \wedge v_y \wedge v_z$, stating that all three of x, y and z have voted.

Fig. 5. Properties satisfied by the implementation at Fig. 3.

$$\textit{3-Election} \quad \leq \quad V_x V_y V_z, \tag{7}$$

is valid. Finally, we use *pKA* reasoning to establish that refinement, with the minimal amount of numerical calculation, and in particular without having to consider all possible execution orders.

First we set out at Fig. 5 some simple properties of the programs V_i's (for $i \in \{x, y, z\}$)—these algebraic rules provide the hypotheses under which the protocol may be proved correct, and collectively they form an algebraic specification.[11]

Whilst the algebraic proof below ensures that the specification (7) is met, when presented with a concrete system (such as Fig. 3) the hypotheses must be verified directly using the probabilistic semantics Fig. 1, to complete the link between the concrete protocol and the abstract algebraic reasoning. The advantage of using this kind of verification rather than building a concrete model of the whole distributed protocol (in general involving many more than just 3 processes) is that all quantitative reasoning at the concrete level becomes entirely localised and can be carried out in small pieces: in some cases it can even be automated [10].

Fortunately for the protocol at Fig. 3, checking the hypotheses is very easy. In (a) for example once V_i has executed, then the guard in V_i makes $(1 + V_i)$ behave as skip. For (b) we use the program- (not Kleene-) algebra of pGCL in a similar style to our earlier proof of the induction step (but dropping the "$\bigoplus$" for neatness): we reason

$$(w := i \ {}_{1/n}\!\oplus \mathsf{skip});\ (w := j \ {}_{1/n+1}\!\oplus \mathsf{skip})$$

$$= \left|\begin{array}{ll} w := i;\ w := j & @(1/n)(1/n{+}1) \\ w := i;\ \mathsf{skip} & @(1/n)(1 - 1/n{+}1) \\ \mathsf{skip};\ w := j & @(1 - 1/n)(1/n{+}1) \\ \mathsf{skip};\ \mathsf{skip} & @(1 - 1/n)(1 - 1/n{+}1) \end{array}\right.$$

$$= \left|\begin{array}{ll} w := j & @(1/n)(1/n{+}1) \\ w := i & @(1/n)(1 - 1/n{+}1) \\ w := j & @(1 - 1/n)(1/n{+}1) \\ \mathsf{skip} & @(1 - 1/n)(1 - 1/n{+}1) \end{array}\right.$$

$$= \left|\begin{array}{ll} w := i & @1/n{+}1 \\ w := j & @1/n{+}1 \\ \mathsf{skip} & @1 - 2/n{+}1, \end{array}\right.$$

which is symmetric in i, j.

[11] Note that the extra "1+" in (a) and (c) are examples of where the stronger hypotheses are needed to allow appeal to the (daggered) rules at Fig. 2.

For (c) we assume first that $v_x = v_y = v_z = \mathit{false}$ initially, and we note that since execution of V_i does not change the variables v_j for $j \neq i$, then $(1+V_x)(1+V_y)(1+V_x)[T] = V_x V_y V_z[T]$. The result now follows for this case since by property Fig. 5(b) the V_i's commute. The other cases where one or more of the v_i's is initially *true*, the argument is the same, once we observe that $(1 + V_i)$ is equivalent to the identity 1 in the case that $v_i = \mathit{false}$ initially.

More generally, approaches for carrying out such small intricate proofs can be done using quantitative program logic described elsewhere [20]. Whilst that logic is not optimised for checking full refinements, in some special cases there do exist practical verification rules for probabilistic refinement checking [10].

Now we have algebraic properties at Fig. 3, we are able to prove entirely within the algebra *pKA* the refinement given by (7) and we note that crucially the proof does not rely on detailed arithmetic calculation at all. Proofs proceed effectively by re-writing subexpressions to equivalent, or weaker ones, with the validity guaranteed by the hypotheses (in this case Fig. 5) or the general *pKA* axiom set at Fig. 2. We illustrate the style with the following calculation.

Lemma 5. *The hypotheses at Fig. 5 imply* (7)*, i.e.* 3-*Election* $\leq V_x V_y V_z$.

Proof. Write V for $(1+V_x)(1+V_y)(1+V_z)$, and reason

$$
\begin{array}{lr}
(V_x+V_y+V_z)V[T] & \\
= V_x V[T]+V_y V[T]+V_z V[T] & \text{Fig. 2(iv,ix)} \\
= V_x(1+V_x)(1+V_y)(1+V_z)[T] + V_y V[T]+V_z V[T] & \text{Definition of } V \\
= V_x(1+V_y)(1+V_z)[T] + V_y V[T]+V_z V[T] & V_x(1+V_x) = V_x, \text{ Fig. 5(a)} \\
\leq V[T]+V_y V[T]+V_z V[T] & V_x \leq (1+V_x); \text{ definition of } V \\
\leq V[T] + V[T] + V[T] & \text{As above and Fig. 5(c left)} \\
= V[T]. &
\end{array}
$$

From this we can appeal to Fig. 2(xiii) and Fig. 5(c right) to deduce that

$$(V_x+V_y+V_z)^* V[T] \quad \leq \quad V[T],$$

but now $1 \leq V$ and so we have after all that the left-hand side is at least $(V_x+V_y+V_z)^*[T]$, from which (7) follows.

In fact this result can be applied more generally to an election of N voters: provided the scheduler allows each voter to make their choice, the election will be fair. Here the scheduler is still able to choose the order, yielding many complicated trace patterns than are possible in the 3-voter election. Unfortunately (7) does not generalise, and in this case we need to prove a slightly weaker result using the more complicated serialisability results dealt with in Section 7 below.

Rather than setting out that algebraic proof here, we discuss those more general serialisation-style theorems in the next section. We examine how to carry out formal proofs of such algebraic properties in Section 5, and apply the results to a much more sophisticated mutual exclusion protocol, of which fairness is a subtle issue.

4. Separation theorems

Voting schemes usually make up only part of a larger protocol, designed to achieve some other goal such as electing a leader. Thus to be useful the protocol must ensure that the winner and losers are somehow notified of the result. Thus a protocol is often divided into two "phases": one for voting, and one for notification.[12] In distributing such phased protocols, typical is the apparent mixing up of the two phases, with the consequence that it is no longer clear that the fairness criterion is met. One of the tasks of the verification exercise is to show that the phases can be separated throughout the system, revealing after all a pattern of straightforward behaviour which can then be more easily analysed.

Separation is the standard technique applicable in such situations, and in this section we extend some standard separation theorems of Cohen [6] to the probabilistic context, so that we may apply them to probabilistic protocol

[12] A *phase* of a program is a fragment of a full program—when represented by a term a, it satisfies all the rules in Fig. 2.

verification. Although the lemmas are somewhat intricate we stress their generality: proved once, they can be used in many applications. We do not attempt to explain the proofs in detail in *pKA* here—we leave that until Section 5, where we discuss proof automation.

Our first results at Lemma 6 consider a basic iteration, generalising loop-invariant rules to allow the body of an iteration to be transformed by passage of a program a.

Lemma 6

$$ac \leq cb \;\Rightarrow\; a^*c \leq cb^* \tag{8}$$

$$a(b+1) \leq ca + d \;\Rightarrow\; ab^* \leq c^*(a + db^*) \tag{9}$$

Proof. For (8) see the Appendix; for (9) see [7, Lemma 2(6)]. □

Note that the weaker commutativity condition of $ab \leq ca + d$ will not do at (9), as $a, b, c \mathrel{\hat{=}} \mathit{flip}$ and $d \mathrel{\hat{=}} \mathsf{magic}$ illustrates. In this case we recall that a^* and b^* both correspond to the program $s := 0 \sqcap 1$, and this is not the same as the corresponding interpretation for c^*a which corresponds to *flip* again.

Lemma 6 implies that with suitable commutativity between phases of a program, an iteration involving the interleaving of the phases may be thought of as executing in two separated pieces. Here, if we think of a and b now representing those two phases, we note that again we need to use a hypothesis $b(1 + a) \leq (1 + a)b$, rather than a weaker $ba \leq ab$.

Lemma 7. $b(1 + a) \leq (1 + a)b \quad\Rightarrow\quad (a + b)^* \leq a^*b^*$.
Proof. See Lemma 2 at *Computer Formalised Proofs* [7]. □

In some cases however we might know that b is standard (has no probability), so that it distributes $+$ from the right: in that case the hypothesis of Lemma 7 is after all equivalent to the simpler $ba \leq ab$.

5. Computer support for *pKA* proofs

One could express *pKA* in the logical framework of one of the several successful proof assistants. But that could easily turn into a major proof formalisation effort on its own: compare the experience of Hurd in a closely related problem, the formalisation of *pGCL* in HOL [12].

Instead we chose the simpler and much more accessible alternative of using the interactive theorem prover KAT-ML developed by Aboul-Hosn and Kozen [3,1]. This tool is designed to help a user to perform interactive equational (and quasi-equational) proofs in the formal system of Kleene algebras with tests (*KAT*), an extension of Kleene algebras with an embedded Boolean subalgebra (see elsewhere for details [14]). A reason for our choosing KAT-ML is that with a relatively simple modification of its code, we could use a new set of axioms for *pKA* instead of the hard-coded original set for *KA*. This did not require any modification of the proof machinery of the original tool, thus saving a big amount of effort. A second reason, and in particular for not doing this development in a logical framework (such as Isabelle, HOL or Coq) is one that the authors of KAT-ML already held: as the intended use of the tool is performing equational reasoning (as opposed to, say, formalising the model theory of *pKA*), the full power and complexity of those other tools is not necessary, and the extra effort required for a user to master enough of those tools' particularities is avoided in favor of a simpler and cleaner development.

KAT-ML is based on the concepts set out in Aboul-Hosn's PhD thesis [2], in which the relationship between proofs in a proof library is just as formal as the steps of those same proofs. While the prover represents the proofs done by the user as λ-terms, the user does not need to deal with this representation and works purely in the KAT syntax. Proofs can be stored in and retrieved from library files, and there's also a facility to create nicely formatted LaTeX versions of the proofs. The prover can be used in a command-line or a graphical interface mode—in particular the interaction involves specifically "focusing" and "unfocusing" on parts of an equation to be proved. The focussed part is rewritten by appealing to the appropriate axioms or previously proved theorems in library files. We set out the axioms and an example proof in the appendix. Regarding the automation provided by this tool, it is able to explore the search tree of a goal proof to a depth indicated in a run-time parameter (inside a configuration file), and this search tries to make

use not just of the axioms for pKA but also of any known theorems already available in the proof environment as it stands at that moment. In practice, we have found this pretty much eliminates the need for the user to do almost all the tedious parts of proofs (in particular the many applications of commutativity and monotonicity), and it takes care of a good amount of non-trivial applications of other axioms and theorems.

Considering all the features just described, and the closeness of pKA to KA as algebraic systems (as discussed in the introduction of the present work), we have chosen to use a modified version of KAT-ML to support our proofs mechanically. The modification consists of replacing the hard-coded set of axioms in the original KAT-ML prover by the corresponding set of pKA axioms shown above at Fig. 2. At this point, the *new* axiom set implies that many of the theorems in the original KAT-ML library are incorrect, and we needed to reconstruct by hand a new library set based on *pKA*. Fortunately many slightly weaker versions of the original theorems were found to remain true for pKA, however often with new and different proofs.

Aside from that our modification left the interface elements and the whole of the prover's engine untouched. The program with its sources and the new proof libraries can be found through the second author's web page [7].

A desirable further modification in the near future would be to make the axiom set loadable from an external file, so both our proofs and the original ones for KAT can be used with the same program. This would also allow the tool to support other variations of Kleene algebra. A limitation of our tool that comes from the original system is the inability to do proofs that involved a variable number of formulas (think for instance of a general distributivity theorem where the number of elements involved is arbitrary, as in Lemma 8 and Lemma 9 below). While in a few cases some suitable introduction of names for such variable-length expressions and the provision of special hypothesis could allow one to carry a proof of such theorem, in general we need more than the tool currently provides. In this respect, logical frameworks could be a good solution: interfacing our modified KAT prover with such a framework would be very useful for that kind of proof.

6. Mutual exclusion with bounded waiting

In this section we describe the mutual exclusion protocol, and discuss how to apply the algebraic approach to it.

> Let $P_1, \ldots P_N$, be N processes that from time to time need to have exclusive access to a shared resource.
>
> The *mutual exclusion problem* is to define a protocol which will ensure both the exclusive access, and the "lockout-free" property, namely that any process needing to access the shared resource will eventually be allowed to access it.
>
> A protocol is said to satisfy the *bounded waiting condition* if, whenever no more than K processes are actively competing for the resource, each has probability at least α/K of obtaining it, for some fixed α (independent of N).[13]

The randomised solution we consider is based on one proposed by Rabin [15]. Processes can coordinate their activities by use of a shared "test-and-set" variable, so that "testing and setting" is an atomic action. The solution assumes an "adversarial scheduler", the mechanism which controls the otherwise autonomous executions of the individual processes. The scheduler chooses nondeterministically between them, and the chosen P_i then may perform a single atomic action, which might include the test and set of the shared variable together with some updates of its own local variables. Whilst the scheduler is not restricted in its choice, it treats the processes fairly in the sense that it must always eventually schedule any particular process.

The broad outline of the protocol is as follows; more details are set out at Fig. 6. Each process executes a program which is split into two phases, one voting, and one notifying. In the voting phase, processes participate in a lottery, each drawing a number; the current winner's lottery number is recorded as part of the shared variable. Processes draw at most once in each voting phase, and the winner is notified when it executes its notification phase. The notification phase begins only when the critical section becomes free.

Our aim is to show that when processes follow the above protocol, the bounded-waiting condition is satisfied. Rabin observed that in a lottery in which tickets are drawn according to (independent) exponential distributions, there is a

[13] Note that this is a much stronger condition than a probability α/N for some constant α, since it is supposed that in practice $K \ll N$.

- *Voting phase.* P_i checks whether it is eligible to vote; if so, it draws a number according to the exponential distribution; if that number is strictly greater than the largest value drawn so far, it sets the shared variable to that value, and keeps a record. If not (has voted already in this round), it skips.
- *Notification phase.* P_i checks whether it is eligible to test (has voted) and, if it is, then checks whether its recorded vote is the same as the current maximum (by examining the shared variable); if it is, it sets itself to be the winner. If P is ineligible, then it just skips.
- *Release of the critical section.* This is indicated by the execution of a program fragment C, which resets a "busy" flag. When C is executed, the critical section becomes free, and processes may begin notification.

These events occur in a single round of the protocol; the verifier of the protocol must ensure that when these program fragments are implemented, they satisfy the algebraic properties set out at Fig. 7.

Fig. 6. The key events in a single round of the mutual exclusion protocol.

probability of at least $2/3$ of a unique winner [15].[14] However that model-based proof cannot be applied directly here, since it assumes (a) that there is no scheduler/probability interaction; (b) that the voting is unbiased between processes, and (c) that the voting may be separated from the notification. In Rabin's original solution, (c) was false, and that in turn implied problems with (a) and (b) as well. We observe however that his highly novel model-based argument still applies provided that voting may be (almost) separated from notification. We shall use an algebraic approach to do exactly that for a modified protocol, which does indeed allow the separation, and is a feature of a later proposal for solving the mutual exclusion problem [15].

7. The probability that a participating process loses

We now show how the lemmas of Section 4 can be applied to the abstract example of Section 6: we compute the probability that a particular process P_0 loses. Writing V_0 and T_0 for the two phases of P_0's protocol, respectively vote and notify (recall Fig. 6), and we introduce abbreviations $\widehat{V} \mathrel{\hat{=}} +_i V_i$ and $\overline{V} \mathrel{\hat{=}} +_{i\neq 0} V_i$ so that $\widehat{V} = V + \overline{V}$; similarly $\widehat{T}$ and $\overline{T}$. The chance that P_0 loses can be computed easily from the following expression

$$(\widehat{V} + \widehat{T} + C)^*\, A_0,$$

where A_0 abstracts the results, preserving only the value of the highest vote (in the case that P_0 drew it or has not yet voted), or that P_0 has lost. In the lemmas below we shall show that this expression is equivalent to a competition between P_0 and a pool of anonymous contestants—overall the competition can be serialised, thus simplifying the calculation necessary to compute the actual chance of P_0's success.

The necessary algebraic properties of the program fragments are set out at Fig. 7, and as a separate analysis the verifier must ensure that the actual code fragments implementing the various phases of the protocol satisfy them. This, as we saw for*3-Election*, is done at a lower conceptual level, in the program algebra of pGCL, rather than in the Kleene algebra; the crucial advantage is that the Kleene-algebraic "pre-analysis" has allowed those code fragments to be treated one-by-one, in isolation. An alternative way to think of Fig. 7 is as an *algebraic specification* for the concrete protocol code.

The next lemma uses separation to show that we can separate the voting from the notification within a single round, with the round effectively ending the voting phase with the execution of the critical section. (Note that we cannot prove

[14] Each process P_i independently chooses $t_i \geq 1$ with probability $1/2^{t_i}$; then with probability $\geq 2/3$ one choice will strictly exceed all others, no matter how many processes chose. (If the choices are bounded, the $2/3$ must be adjusted.)

(1) *Any vote commutes with any notification event*: $V_i T_j = T_j V_i$.

(2) *Notification occurs when the critical section is free*: $T_i(C+1) \leq (C+1)T_i$.

(3) *Voting occurs when the critical section is busy*: $C(V_i+1) \leq (V_i+1)C$.

(4) *The order of voting does not affect P_0's success*: $V_0 A_0(\overline{V}A_0 + 1) \leq (\overline{V}A_0 + 1)V_0 A_0$.

Recall that $a \leq b$ means that each possible outcome of a occurs in b also and with a probability at least as high; the aim is that b is simpler than a; and thus analysing b for "bad" properties provides a conservative estimate for the occurrence of those bad properties in a. We note also that A_0 satisfies properties of (standard) abstraction functions: it distributes $+$, it is idempotent so that $A_0A_0 = A_0$, and it acts as an abstraction function: $A_0(V_0 + \overline{V})A_0 = (V_0 + \overline{V})A_0$.

Fig. 7. Algebraic properties of the system.

Lemma 8 and Lemma 9 completely in KAT-ML as they contain parameters which cannot be expressed in that system; the proofs below are done by hand where necessary.)

Lemma 8. *We may assume all voting to come before the critical section is released, and all notification to come after*:

$$(\widehat{V} + \widehat{T} + C)^* \quad \leq \quad \widehat{V}^*\, C^*\, \widehat{T}^*.$$

Proof. From Fig. 7 we can show

$$\widehat{T}\,(\widehat{V} + C + 1) \quad \leq \quad (\widehat{V} + C + 1)\,\widehat{T}, \quad \text{and}$$
$$C(\widehat{V} + 1) \quad \leq \quad (\widehat{V} + 1)C$$

(since $\widehat{T}$ has a standard denotation, so distributes $+$). These now form the general hypotheses for Lemma 8 at *Computer Formalised Proofs* [7], from which our result follows. □

Next we may consider the voting to occur in an orderly manner in which the selected process P_0 votes last, with the other processes effectively acting as a pool of anonymous opponents who collectively attempt to lower the chance of its winning—this is the fact allowing us to use the model-based observation of Rabin to compute a lower bound on the chance that P_0 wins.

Lemma 9. *Participant P_0's chance of losing is bounded above by that chance in a schedule in which it votes last*:

$$\widehat{V}^* A_0 \quad \leq \quad \overline{V A_0}^* (V_0 A_0)^* A_0.$$

Proof. The proof can be found at Lemma 9 at *Computer Formalised Proofs* [7]. The hypotheses may be deduced from Fig. 7(c) and the properties of A_0 listed there.[15] □

With Lemma 9, we have finally established the goal of serialising a single round of Rabin's (revised) protocol, thus justifying Rabin's original intuitions [15] that participants' votes can be considered to be carried out independently of eachother. However that justification relies on the separation of the voting and notification phases, each controlled by the release of the critical section—that crucial separation, which emerged in a later protocol [15], was not present in Rabin's original proposal.

To complete the proof of the fairness criterion a detailed numerical proof is required, set out elsewhere [12]—however we note here that the proof assumes that the round can be separated as above. Thus Lemma 9 plays the crucial role in establishing the hypotheses required for the results of the numerical proof to apply to Rabin's protocol.

[15] Note that in KAT-ML it is not possible to state the general distributivity (of A_0) through $+$ as a hypothesis; any appeal to distributivity must be stated explicitly relative to specific expressions.

The calculation above is based on the assumption that P is eligible to vote when it is first scheduled in a round. In Rabin's implementation, the mechanism for testing eligibility uses a round number as part of the shared variable, and after a process votes, it sets a local variable to the same value as the round number recorded by the shared variable. By this means the process is prevented from voting more than once in any round. In the case that the round number is unbounded, P will indeed be eligible to vote the first time it is scheduled. However one of Rabin's intentions was to restrict the size of the shared variable, and in particular the round number. His observation was that round numbers may be reused provided they are chosen *randomly* at the start of the round, and that the *scheduler cannot see the result* when it decides which process to schedule. In the next section we discuss the implications of this assumption on $\mathcal{LS}$ and *pKA*.

8. Secrecy and its algebraic characterisation: discussion

The actual behaviour of Rabin's protocol includes *probabilistically* setting the round number, which we denote R and which makes the protocol in fact

$$(R(\widehat{V} + \widehat{T} + C)^*)^*, \tag{10}$$

where the outer $*$ means that the single round is repeated some number of times.

The problem is that the interpretation in $\mathcal{LS}$ assumes that the value chosen by R is observable by all, in particular by the adversarial scheduler, that latter implying that the scheduler can use the value during voting to determine whether to schedule P. In a multi-round scenario, that would in turn allow the policy that P is scheduled *only* when its just-selected round variable is (accidentally) the same as the current global round: while satisfying fairness (since that equality happens infinitely often with probability one), it would nevertheless allow P to be scheduled only when it cannot possibly win (in fact will not even be allowed to vote). Thus, in this case we would find that P's ineligibility to vote would be correlated with his being selected to participate in any round—just as in the adversary in the simple voting mechanism could correlate his choice of which process to schedule, based on the result of a previous probabilistic choice.

Clearly that strategy must be prevented (if the algorithm is to be correct!)—and unfortunately in this case it must be prevented by the explicit assumption that the scheduler cannot see the value set by R. However the scheduler formalised using the nondeterminism given by Definition 1 implies that it *can* see the result of previous probabilistic outcomes—that is precisely the behaviour illustrated by Fig. 4. Thus we need a rather more complicated model to support algebraic characterisations for "cannot see", and we end this section by discussing what that would be.

The following (sketched) description of a model $\mathcal{QS}$ [19, Key QMSRM]—necessarily more detailed than $\mathcal{LS}$—is able to model cases where probabilistic outcomes cannot be seen by subsequent demonic choice. The idea (based on "non-interference" in security) is to separate the state into *visible* and *hidden* parts, the latter not accessible directly by demonic choice. The state s is now a pair (v, h) where v, like s, is given some conventional type but h now has type *distribution* over some conventional type. The $\mathcal{QS}$ model is effectively the $\mathcal{LS}$ model built over this more detailed foundation.[16]

For example, if a sets the hidden h probabilistically to 0 or 1 then (for some p) in the $\mathcal{QS}$ model a denotes

Hidden resolution of probability $\quad (v, h) \quad \overset{a}{\mapsto} \quad \{\, (v,\ (0\ {}_p\!\oplus\ 1)\ )\,\}$.[17]

In contrast, if b sets the visible v similarly we'd have b denoting

Visible resolution of probability $\quad (v, h) \quad \overset{b}{\mapsto} \quad \{\, (0, h)\ {}_{1/2}\!\oplus\ (1, h)\,\}$.

The crucial difference between a and b above is in their respective interactions with subsequent nondeterminism; for we find

$$\begin{array}{lrcl} & a(c+d) & = & ac + ad \\ \text{but in general} & b(c+d) & \neq & bc + bd, \end{array}$$

[16] Thus we have "distributions over values-and-distributions" so that the type of a program in $\mathcal{QS}$ is $(V \times \overline{H}) \to \mathbb{P}\,(\overline{V \times \overline{H}})^{\top}$, that is $\mathcal{LS}$ where $S = V \times \overline{H}$.

[17] Strictly speaking we should write $\delta_0\ {}_p\!\oplus\ \delta_1$.

because in the a case the nondeterminism between c and d "cannot see" the probability hidden in h. In the b case, the probability (in v) is not hidden.[18]

A second effect of hidden probability is that tests are no longer necessarily "read-only". For example if t denotes the test $[h = 0]$ then we would have (after a say)

$$(v,\ (0\ {}_p\!\oplus\ 1)\ ) \quad \stackrel{t}{\mapsto} \quad \{(v, 0)\ {}_p\!\oplus\ \mathsf{magic}\}$$

where the test, by its access to h, has revealed the probability that was formerly hidden and, in doing so, has changed the state (in what could be called a particularly subtle way—which is precisely the problem when dealing with these issues informally!)

In fact this state-changing property gives us an algebraic characterisation of observability.

Definition 10. Observability; resolution.

For any program a and test t we say that "t is known after a" just when

$$a(t + \neg t) \ = \ a. \tag{11}$$

As a special case, we say that "t is known" just when $t + \neg t = 1$.

Say that Program a "contains no visible probability" just when for all programs b, c we have

$$a(b + c) \quad = \quad ab + ac.$$

Thus the distributivity through $+$ in Definition 10 expresses the adversary's ignorance in the case that a contains hidden probabilistic choice. If instead the choice were visible, then the $+$-distribution would fail: if occurring first it could not see the probabilistic choice [19] whereas, if occurring second, it could.

Secrecy for the randomised round number

We illustrate the above by returning to mutual exclusion. Interpret R as the random selection of a local round number rn (as suggested above), and consider the probability that the adversarial scheduler can guess the outcome. For example, if the adversary may guess the round number (here assumed to take only two values) with probability 1 during the voting phase, according to Definition 10 we would have

$$R\ \widehat{V}^*([rn = 0] + [rn = 1])\ \mathsf{chaos} \ = \ \mathsf{chaos},$$

(because $[rn = 0] + [rn = 1]$ would be skip).[20] But since

$$(\widehat{V} + 1)([rn = 0] + [rn = 1]) \ = \ ([rn = 0] + [rn = 1])(\widehat{V} + 1), \tag{12}$$

we may reason otherwise:

$$\begin{array}{ll} R\widehat{V}^*([rn = 0] + [rn = 1])\mathsf{chaos} & \\ = R([rn = 0] + [rn = 1])(\widehat{V})^*\mathsf{chaos} & \text{(9) and (8); see below.} \\ = R[rn = 0]\mathsf{chaos} + R[rn = 1]\mathsf{chaos}. & \text{Definition 10 and (11)} \end{array}$$

For the "see below", we note that the equality at (12) can be reformulated as two inequalities. Then, with the interpretation $d \mathrel{\hat{=}} \mathsf{magic}$, $b, a \mathrel{\hat{=}} (\widehat{V} + 1)$ and $c \mathrel{\hat{=}} [rn = 0] + [rn = 1]$ we can apply separately (9) and (8) at Lemma 6, to give the required equality.

Now, back in the model, we can compute $R[rn = 0]\mathsf{chaos} + R[rn = 1]\mathsf{chaos} = \mathsf{magic}\ {}_{1/2}\!\oplus\ \mathsf{chaos}$, and deduce that the chance that the scheduler may guess the round number is at most $1/2$, and not 1 at all.

[18] See the example set out below Theorem 4 for an illustration of the non equality of $b(c + d)$ and $bc + bd$, in the context where nondeterministic choice is always sensitive to the outcome of preceding probabilistic choices.

[19] Here it cannot see it because it has not yet happened, not because it is hidden.

[20] Here we are abusing notation, by using program syntax directly in algebraic expressions.

We end by noting that this model has been worked out in detail for verifying secrecy-style properties with the probabilities abstracted [23].

9. Conclusions and other work

Rabin's probabilistic solution to the mutual exclusion problem with bounded waiting is particularly apt for demonstrating the difficulties of verifying probabilistic protocols, as the original solution contained a particularly subtle flaw [28]. The use of *pKA* makes it clear what assumptions need to be checked relating to the individual process code and the interaction with the scheduler, and moreover a model-based verification of a complex distributed architecture is reduced to checking the appropriate hypotheses are satisfied. Our decision to introduce the models separately stems from $\mathcal{QS}$'s complexity to $\mathcal{LS}$, and the fact that in many protocols $\mathcal{LS}$ is enough. The nice algebraic characterisations of hidden and visible state, may suggest that $\mathcal{QS}$ may support a logic for probabilities and ignorance in the refinement context, though that remains an interesting research.

A number of topics for investigation are suggested by our experiments in automating the proofs. For example the KAT-ML tool does not handle proofs with generalised choice, which are often features of parameterised proofs. It would also be interesting to explore practical techniques for verifying the hypotheses at the concrete level. The problem is essentially one of verifying general program refinements in the probabilistic context—whilst the theory of probabilistic refinement is well-understood, there are very few examples of automated proof assistants for its verification [10], and even in those cases they apply to a restricted class of refinements.

A more practical approach could involve techniques used in probabilistic model checking, but should be a significantly simpler problem, as they would only need be applied to $*$-free programs.

Others have investigated instances of Rabin's algorithm using model checking [25]; there are also logics for "probability-one properties" [16], and models for investigating the interaction of probability, knowledge and adversaries [8].

Appendix

A. *pKA* axiom list for KAT prover formalisation

The axiom list below is that used within the adaptation of the KAT-ML prover [3]. Note that the sequential composition ab is expressed as $\mathsf{a} \cdot \mathsf{b}$; moreover associativity of $+$ and $\cdot$ is handled automatically and implicitly by the KAT prover

(ref=)	$x = x$
(sym=)	$x = y \;\Rightarrow\; y = x$
(trans=)	$x = y \;\Rightarrow\; y = z \;\Rightarrow\; x = z$
(cong+R)	$x = y \;\Rightarrow\; x + z = y + z$
(cong.L)	$y = z \;\Rightarrow\; x \cdot y = x \cdot z$
(cong.R)	$x = y \;\Rightarrow\; x \cdot z = y \cdot z$
(cong*)	$x = y \;\Rightarrow\; x^* = y^*$
(<intro)	$x + y = y \;\Rightarrow\; x \leq y$
(<elim)	$x \leq y \;\Rightarrow\; x + y = y$
(commut+)	$x + y = y + x$
(id+L)	$0 + x = x$
(idemp+)	$x + x = x$
(id.L)	$1 \cdot x = x$
(id.R)	$x \cdot 1 = x$
(annihL)	$0 \cdot x = 0$
(annihR)	$x \cdot 0 = 0$
(distrL)	$x \cdot y + x \cdot z \leq x \cdot (y + z)$
(distrR)	$(x + y) \cdot z = x \cdot z + y \cdot z$
(unwindL)	$x^* = 1 + x \cdot x^*$
(*L)	$x \cdot (y + 1) \leq x \;\Rightarrow\; x \cdot y^* = x$
(*R)	$x \cdot y \leq y \;\Rightarrow\; x^* \cdot y = y$

B. Auxiliary theorems

The theorems below are all proved using the above equational axioms.

(trans<)	$x < y \Rightarrow y < z \Rightarrow x < z$
(add∗R)	$x < x \cdot y^*$
(=<)	$x = y \Rightarrow x < y$
(supR)	$y < x + y$
(mono.R)	$x < y \Rightarrow x \cdot z < y \cdot z$
(mono.L)	$y < z \Rightarrow x \cdot y < x \cdot z$

C. An example KAT-ML style automated proof for *pKA*

The KAT-ML tool outputs the results of a completed interactive proof in a "pretty-printed" style, an example of which is set out here. All the other automated proofs may be accessed at [7].

Note that the proof obligations are given in a list—for example both (13) and (14) below are the result of appealing to trans<, on the right hand side of (12), and both goals must be discharged with further appeal to the axioms or other already proved properties.

Theorem 11 (Lemma 6(9))

$$a \cdot c \leq c \cdot b \quad \Rightarrow \quad a^* \cdot c \leq c \cdot b^*. \tag{C.1}$$

By trans<, it suffices to show that

$$a^* \cdot c \leq a^* \cdot c \cdot b^* \tag{C.2}$$

$$a^* \cdot c \cdot b^* \leq c \cdot b^* \tag{C.3}$$

Consider (C.2). By add*R, we have what we need.

Consider (C.3). By =<, it suffices to show that

$$a^* \cdot c \cdot b^* = c \cdot b^* \tag{C.4}$$

Consider (C.4). By *R, it suffices to show that

$$a \cdot c \cdot b^* \leq c \cdot b^* \tag{C.5}$$

Consider (C.5). By trans<, it suffices to show that

$$a \cdot c \cdot b^* \leq c \cdot b \cdot b^* \tag{C.6}$$

$$c \cdot b \cdot b^* \leq c \cdot b^* \tag{C.7}$$

Consider (C.6). By mono.R, it suffices to show that

$$a \cdot c \leq c \cdot b \tag{C.8}$$

Consider (C.8). By (C.1), we have what we need.

Consider (C.7). By mono.L, it suffices to show that

$$b \cdot b^* \leq b^* \tag{C.9}$$

Consider (C.9). By trans$<$, it suffices to show that

$$b \cdot b^* \leq 1 + b \cdot b^* \tag{C.10}$$
$$1 + b \cdot b^* \leq b^* \tag{C.11}$$

Consider (C.10). By supR, we have what we need.

Consider (C.11). By $=<$, it suffices to show that

$$1 + b \cdot b^* = b^* \tag{C.12}$$

Consider (C.12). By sym=, it suffices to show that

$$b^* = 1 + b \cdot b^* \tag{C.13}$$

Consider (C.13). By unwindL, we have what we need. □

References

[1] KAT-ML project homepage. <www.cs.cornell.edu/Projects/KAT/>.
[2] Kamal Aboul-Hosn, A proof-theoretic approach to mathematical knowledge management, Ph.D. thesis, Department of Computer Science, Cornell University, 2007.
[3] Kamal Aboul-Hosn, Dexter Kozen, KAT-ML: an interactive theorem prover for Kleene algebra with tests, J. Appl. Non-Classical Logics, 1 (2006).
[4] R.-J.R. Back, J. von Wright, Refinement Calculus: A Systematic Introduction, Springer-Verlag, 1998.
[5] O. Celiku, Mechanisation of dually anelgic and probabilistic programs, Ph.D. thesis, TUVCS, 2006.
[6] E. Cohen, Separation and reduction, Mathematics of Program Construction, 5th International Conference, July, LNCS, vol. 1837, Springer-Verlag, 2000, pp. 45–59.
[7] Carlos Gonzalia, Computer formalised pKA proofs. <www.ics.mq.edu.au/∼carlos>.
[8] J. Halpern, M. Tuttle, Knowledge, probabilities and adversaries, JACM 40 (4) (1993) 917–962.
[9] Jifeng He, K. Seidel, A.K. McIver, Probabilistic models for the guarded command language, Sci. Comput. Programming 28 (1997) 171–192., Available at [19, key HSM95].
[10] T.S. Hoang, C.C. Morgan, A. McIver, K.A. Robinson, Z.D. Jin, Refinement in probabilistic b: Foundation and case study, in: H. Treharne, S. Schneider (Eds.), Proceedings of ZB 2005, LNCS, vol. 3455, Springer, 2005, pp. 252–273.
[11] Joe Hurd, A formal approach to probabilistic termination, in: Víctor A. Carreño, César A. Muñoz, Sofiène Tahar (Eds.), 15th International Conference on Theorem Proving in Higher Order Logics: TPHOLs, Hampton, VA, August, LNCS, vol. 2410, Springer, 2002, pp. 230–245. www.cl.cam.ac.uk/∼jeh1004/research/papers.
[12] Joe Hurd, A.K. McIver, C.C. Morgan, Probabilistic guarded commands mechanised in HOL, Theor. Comput. Sci. (2005) 96–112.
[13] C. Jones, Probabilistic nondeterminism, Monograph ECS-LFCS-90-105, Ph.D. thesis, Edinburgh University, 1990.
[14] Dexter Kozen, Kleene algebra with tests, ACM Trans. Programming Lang. Syst. 19 (3) (1997) 427–443.
[15] Eyal Kushilevitz, M.O. Rabin, Randomized mutual exclusion algorithms revisited, in: Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing, 1992, pp. 275–283.
[16] D. Lehmann, S. Shelah, Reasoning with time and chance, Inform. and Control 53 (3) (1982) 165–198.
[17] A. McIver, E. Cohen, C. Morgan, Using probabilistic Kleene algebra pKA for protocol verification, in: G. Struth, R. Schmidt (Eds.), Proc the 9th International Conference on Relational Methods in Computer Science, LNCS, vol. 4136, Springer, 2006.
[18] A. McIver, T. Weber, Towards automated proof support for probabilistic distributed systems, Proceedings of Logic for Programs and Automated Reasoning, LNAI, vol. 3835, Springer-Verlag, 2005.
[19] A.K. McIver, C.C. Morgan, J.W. Sanders, K. Seidel, Probabilistic Systems Group: Collected Reports. <http://web.comlab.ox.ac.uk/oucl/research/areas/probs>.
[20] Annabelle McIver, Carroll Morgan, Abstraction, Refinement and Proof for Probabilistic Systems, Technical Monographs in Computer Science, Springer-Verlag, New York, 2004.
[21] B. Möller, Lazy Kleene algebra, in: Dexter Kozen, Carron Shankland (Eds.), Mathematics of Program Construction, LNCS, vol. 3125, Springer, 2004, pp. 252–273.
[22] C. Morgan, Private communication, The Lamington model: a probabilistic model with miracles, 1995.
[23] Carroll Morgan, The Shadow Knows: Refinement of ignorance in sequential programs, in: Math Prog Construction, 2006, pp. 359–378.
[24] C.C. Morgan, A.K. McIver, K. Seidel, Probabilistic predicate transformers, ACM Trans. Programming Lang. Syst. 18 (3) (1996) 325–353., doi:acm.org/10.1145/229542.229547.
[25] PRISM, Probabilistic symbolic model checker, www.cs.bham.ac.uk/∼dxp/prism.

[26] M.O. Rabin, N-process mutual exclusion with bounded waiting by $4 \log 2n$-valued shared variable, J. Comput. Syst. Sci. 25 (1) (1982) 66–75.
[27] J.R. Rao, Building on the UNITY Experience: compositionality, fairness and probability in parallelism, Ph.D. thesis, University of Texas at Austin, 1992.
[28] I. Saias, Proving probabilistic correctness statements: the case of Rabin's algorithm for mutual exclusion, in: Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing, 1992.
[29] Roberto Segala, Modeling and verification of randomized distributed real-time systems, Ph.D. thesis, MIT, 1995.
[30] T. Takai, H. Furusawa, Monodic tree kleene algebra, in: G. Struth, R. Schmidt (Eds.), Proceedings of the 9th International Conference on Relational Methods in Computer Science, LNCS, vol. 4136, Springer, 2006.