Using Probabilistic Kleene Algebra *pKA* for Protocol Verification *

AK McIver [a,1,*] C Gonzalia [a,1] E Cohen [b] CC Morgan [c,1]

[a] *Dept. Comp. Sci., Macquarie University, NSW 1209 Australia*

[b] *Microsoft, US*

[c] *Dept. Comp. Sci., University of NSW, NSW 2052 Australia*

Abstract

We propose a method for verification of probabilistic distributed systems in which a variation of Kozen's Kleene Algebra with Tests [11] is used to take account of the well-known interaction of probability and "adversarial" scheduling [17].

We describe *pKA*, a probabilistic Kleene-style algebra, based on a widely accepted model of probabilistic/demonic computation [7,25,17]. Our technical aim is to express probabilistic versions of Cohen's *separation theorems*[4].

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* [12] — is one where verification problems have already occurred: the original presentation was later shown to have subtle flaws [24]. 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.

Finally we show how the algebraic proofs for these theorems can be automated using a modification of Kozen and Aboul-Hosn's KAT-ML [3].

Key words: Kleene algebra, probabilistic systems, probabilistic verification

---

* This is an extension of [14].

* Corresponding author
Email address: anabel@ics.mq.edu.au (AK McIver).

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

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 [22], the source of the error may be very difficult to locate.

One approach to the verification problem is probabilistic model checking, 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 [17,9], 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 proof as 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 [10]. 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 describe a program algebra *pKA* introduced elsewhere [15] in which standard *Kleene algebra* [11] has been adapted to reflect the interaction of probabilistic assignments with nondeterminism, where the latter is used to model the indeterminable actions of the scheduler. Standard (i.e. non-probabilistic) Kleene algebra his been used effectively to verify some non-trivial distributed protocols [4], 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* prevents certain distributions of nondeterminism, just in those cases where the interaction of the probability and nondeterminism is an

issue [25,7,17]. That distribution failure however removes some conventional axioms on which familiar techniques depend: and so we must replace 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 [22,12], whose original formulation was found to contain some subtle flaws [24] 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 are invisible to the adversary — cannot be supported by the usual model for probabilistic systems. 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. Such is the case for 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 (Sec. 2), including a generalisation of Cohen's work on separation [4] for probabilistic distributed systems using *pKA* (Sec. 4);
(2) A "tutorial-style" description of how the algebraic approach may be applied in probabilistic verification of a simple voting scheme (Sec. 3);
(3) Application of the general separation results to Rabin's solution to distributed mutual exclusion with bounded waiting (Sec. 7);
(4) A description of how the algebraic proofs may be automated in an adapted version of Kozen and Aboul-Hosn's KAT-ML system [3] (Sec. 5);

We end in Sec. 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. The $(p, 1{-}p)$-weighted average of distributions d and d' is denoted $d\ {}_p\!\oplus\ d'$. If K is a subset, and d a distribution, we write $d.K$ for $\sum_{s \in K} d.s$. 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.

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 [17] though, depending on the particular application, the conditions imposed on the subsets of (final) probability distributions can vary [20,7]. 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

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

In contrast a program that simulates a possible bias favouring 0 of at most 2/3 is modelled by a nondeterministic choice delimiting a range of behaviours:

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

and in the semantics (given below) its result set is represented by the *set* of distributions defined by the two specified probabilistic choices at (2).

In setting out the details, we follow Morgan et al. [20] and take a domain theoretical approach, restricting the result sets of the semantic functions ac-

cording to an underlying order on the state space. We take a 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.

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* $\sqsubseteq$ *on* $S^\top$ *so that*

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

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 [17]. By doing so the semantics accounts for specific features of probabilistic programs. In this case we impose *up-closure* (the inclusion of all $\sqsubseteq$-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 [20]). 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" [2]) is also accounted for (implied by Cauchy closure). A further bonus is that (as usual) 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* $\mathcal{L}S$ *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* $\sqsubseteq$. *All programs are* $\top$*-preserving (mapping* $\top$ *to* $\{\delta_\top\}$*). The order between programs is defined*

$$\mathit{Prog} \sqsubseteq \mathit{Prog}' \quad \textit{iff} \quad (\forall s \colon S \cdot \mathit{Prog}.s \supseteq \mathit{Prog}'.s) \ .$$

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

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

[2] 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.

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

Skip	$\mathsf{skip}.s$	$\hat{=}$	$\lceil\{\delta_s\}\rceil$,
Miracle	$\mathsf{magic}\ .s$	$\hat{=}$	$\{\delta_\top\}$,
Chaos	$\mathsf{chaos}_K.s$	$\hat{=}$	$\mathbb{P}\overline{K^\top}$
Composition	$(Prog; Prog').s$	$\hat{=}$	$\{\sum_{u:\, S^\top}(d.u) \times d'_u \mid d \in Prog.s; d'_u \in Prog'.u\}$,
Choice	$(\mathit{if}\ B\ \mathit{then}\ Prog\ \mathit{else}\ Prog').s$	$\hat{=}$	$\mathit{if}\ B.s,\ \mathit{then}\ Prog.s,\ \mathit{otherwise}\ Prog'.s$
Probability	$(Prog\ {_p\oplus}\ Prog').s$	$\hat{=}$	$\{d\ {_p\oplus}\ d' \mid d \in r.s; d' \in r'.s\}$,
Nondeterminism	$(Prog\ \sqcap\ Prog').s$	$\hat{=}$	$\lceil\{d \mid d \in (Prog.s \cup Prog'.s)\}\rceil$,
Iteration	$Prog^*$	$\hat{=}$	$(\nu X \cdot Prog; X\ \sqcap\ 1)$.

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 *Prog* and *Prog'*, 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 $\sqsubseteq$-program-to-program function $\lambda X \cdot (Prog; X\ \ \sqcap\ \ 1)$. All programs map $\top$ to $\{\delta_\top\}$.

Fig. 1. Mathematical operators on the space of programs [17].

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 takes the weighted average of the distributions of its operands, and the nondeterminism operator takes their union (with closure).

Iteration is the most intricate of the operations — operationally $Prog^*$ represents the program that can execute *Prog* an arbitrary finite number of times. In the probabilistic context, as well as generating the results of all "finite iterations" of $(Prog \sqcap \mathsf{skip})$ (*viz*, a finite number of compositions of $(Prog \sqcap \mathsf{skip})$), 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. To illustrate, we consider

$$halfFlip \quad \hat{=} \quad \mathsf{if}\ (s = 0)\ \mathsf{then}\ \mathit{flip}\ \mathsf{else}\ \mathsf{skip}\ , \qquad (3)$$

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 the distribution $\delta_0/2^n + (1-1/2^n)\delta_1$. Cauchy Closure implies the result distribution must contain δ_1 as well, because $\delta_0/2^n + (1-1/2^n)\delta_1$ converges to that point distribution as n approaches infinity.

We shall repeatedly make use of *tests*, defined as follows. Given a predicate B over the state s, 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 satisfy the following properties.

- $\mathsf{skip} \sqsubseteq [B]$, meaning that the identity is refined by a test.
- $\mathit{Prog}\ ;\ [B]$ determines the *greatest probability* that *Prog* may establish B. For example if *Prog* is the program *biasFlip* at (2), then $\mathit{biasFlip}\ ;\ [s=0]$ is

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

  a program whose probability of not blocking (2/3) is the maximum probability that *biasFlip* establishes $s = 0$.
- Similarly, $\mathit{Prog}\ ;\ [B]\ ;\ \mathsf{chaos}_K\ =\ \mathsf{magic}\ {}_{p_s}{\oplus}\ \mathsf{chaos}_K$, where $(1-p_s)$ 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.
- If *Prog* contains no probabilistic choice, then *Prog* distributes $\sqcap$, 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. That is the topic of the next section.

2.1 Mapping pKA into $\mathcal{L}S$

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 shall also use *tests*, whose denotations are programs of the kind (4). We normally denote a test by t, and for us its complement is $\neg t$.

The next definition gives an interpretation of pKA in $\mathcal{L}S$.

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

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

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

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

Theorem 4 *([15]) Let $[\![\cdot]\!]$ 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 $[\![b]\!] \sqsubseteq [\![a]\!]$.*

To see why we cannot have equality at (†) in Fig.2, consider the expressions $a(b+c)$ and $ab+ac$, and an interpretation where a is *flip* at (1), and b is skip and c is $s := 1{-}s$. In this case in the interpretation of $a(b+c)$, the demon (at $+$) is free to make his selection after the probabilistic choice in a has been resolved, and for example could arrange to set the final state to $s = 0$ with probability 1, since if a sets it to 0 then the demon chooses to execute b, and if a sets it to 1, the demon may reverse it by executing c. On the other hand, in $ab + ac$, the demon must choose which of ab or ac to execute before the probability in a has been resolved, and either way there is a chance of at least 1/2 that the final state is 1. (The fact that distribution fails says 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 rule, $ab \leq a \Rightarrow ab^* = a$, is unsound for the interpretation in $\mathcal{L}S$, again due to the interaction of probability and nondeterminism. Consider, for example, the interpretation where each of a, b and c represent the *flip* defined at (1) above. We may prove directly that $\mathit{flip}\ ;\ \mathit{flip}^* = s := 0 \sqcap s := 1$, i.e. $\mathit{flip}\ ;\ \mathit{flip}^* \neq \mathit{flip}$ in spite of the fact that $\mathit{flip}\ ;\ \mathit{flip} = \mathit{flip}$. To see why, we note that from Def. 3 the Kleene-star is interpreted as an iteration which may stop at any time. In this case, if a result $s = 1$ is required, then *flip* executes for as long as necessary (probability theory ensures that $s = 1$ will eventually be satisfied). On the other hand if $s = 0$ is required then that result too may be guaranteed eventually by executing *flip* long enough. To prevent an incorrect conclusion in this case, we use instead the sound rule (‡) (for which the antecedent fails). Indeed the effect of the $(1 + \cdot)$ in 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.

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

Fig. 2. Rules of Probabilistic Kleene algebra, *pKA*[15].

pKA purposefully treats probabilistic choice implicitly, and it is only the failure of the equality at (†) which suggests that the interpretation may include probability: in fact it is this property that characterises probabilistic-like models, separating them from those which contain only pure demonic nondeterminism. Note in the case that the interpretation is standard — where probabilities are not present in a — then the distribution goes through as usual. 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* [25]. 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), 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 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 $ab = ba$ (program fragments a and b 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 $\mathcal{L}S$ 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 Sec. 6 below we describe how the method can be used for a much more intricate protocol.

3 A simple voting scheme

Voting can be used to resolve conflict situations which often arise in distributed systems, such as leader election, or attempts to gain exclusive access to a critical section. Probability can offer an attractive implemention mechanism, and in this section we show how the algebraic approach can be used to reduce the overall resources required in the related verification task, namely that despite the adversarial scheduling, voting protocols using probability can still be impartial. Consider the following typical voting problem.

> 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, in which an adversarial scheduler "moderates" which of the processes is allowed to participate. The protocol must be designed to meet the following fairness criterion:*for any pair of processes y and z, the chance that z wins is no greater than the chance that y wins (and vice versa), given any competition in which they both participate.*

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 [23]. 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 set $\mathcal{X}$ of processes — is specified using the Kleene star over their generalised sum [3],

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

where the adversarial choice is modelled by the nondeterministic choice. In program execution terms, we are saying that the each process may execute its

[3] 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 $\mathcal{I}$.

voting program “for a while”.[4]

The intuition behind this staightforward 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 the *overall* result of $V_x V_y$ (first x votes and then y) with $n = 0$ initially, is that w is set to x or y each with probability 1/2. To see that, first consider that when $n = 0$ initially, w is set to x with probability 1 (after execution of V_x); next (with n now 1), when V_y executes, w is overwritten by y with probability 1/2 (n is incremented to 2 just before the actual vote is cast). Similarly it can be checked that after n votes have been cast — *in whatever order* — each participant has probability $1/n$ of being the winner, provided they vote independently one after another. Thus this informal argument seems to ensure the pairwise fairness criterion stated above.

Unfortunately that argument is invalid in the context of adversarial scheduling (and indeed it does not take it into account!) In the very liberal interpretation at (5), recall 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.

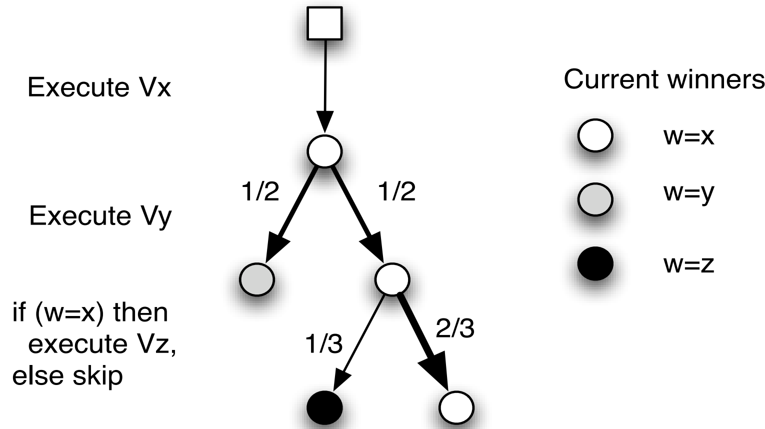
$V_x : (\mathsf{if}\ \neg v_x) \rightarrow$	*If x has not yet participated ...*
$\quad v_x \mathrel{\hat=} \mathit{true}$;	*record his participation ...*
$\quad n := n + 1$;	*increase the participation count ...*
$\quad w := x\ {}_{1/n}\!\oplus\ \mathsf{skip}$	*and now flip to win.*

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

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

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

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 $(V_x + V_y + V_z)^*$ chooses first to execute V_x, then V_y, and then V_z, but 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. [5]



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 z, but *only* in the case that y has lost.

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. [6] 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. If we did have the distributivity, then we would easily be able to show that *Election* is equivalent

[5] 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.
[6] It turns out however that "oblivious scheduling" does have a neat algebraic characterisation as a distribution law involving $+$, and we discuss this later in Sec. 8.

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 [7]), 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.

(a) Candidates may only vote once: $V_i(1+V_i) = V_i, \quad i \in \{x,y,z\}$.

(b) Voting is independent: $V_iV_j = V_jV_i$, for all $i \in \{x,y,z\}$.

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

$$(1{+}V_x)(1{+}V_y)(1{+}V_x)[T] \quad \leq \quad (1{+}V_{\sigma(x)})(1{+}V_{\sigma(y)})(1{+}V_{\sigma(z)})[T] \quad \leq \quad V_xV_yV_z \ ,$$

where σ is any bijection on $\{x,y,z\}$.

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.

One simple way to avoid the scheduler deciding to exclude a process is to enforce the condition that all three must participate. We do this with the test $T \hat{=} v_x \wedge v_y \wedge v_z$, 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 intuitively excludes the schedule depicted in Fig.4.

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

$$\textit{3-Election} \quad \leq \quad V_xV_yV_z \ , \tag{7}$$

is valid. We use *pKA* reasoning to show that it is, with the minimal amount of numerical calculation, and in particular without having to consider all possible

[7] The problems with such informal specifications, especially in the probabilistic context, have been pointed out by Saias [24].

execution orders. Here's how.

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 form an algebraic specification.

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 distributed protocol (in general involving many more than just 3 processes) is that all quantitative reasoning at the concrete level becomes entirely localised, and in some cases can even be automated [8].

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. Similarly for (b) since the result of the vote depends only on w and n, a small calculation shows that the individual votes commute. To see that for example, suppose $n = N$, and $w = i$, and suppose also that $i \notin \{x, y\}$ — we write (N, i) for this state. Next we calculate the result of executing $V_x V_y$ (first x votes, and then y), based on the semantics at Fig.1. Note that in the calculation we use the informal notation $(N, i) \stackrel{a}{\mapsto} (N', i')\ {}_p\!\oplus (N'', i'')$ to mean that, in executing the program denoted by "a", the state (N, i) is transformed to the probability distribution over final states (N', i') and (N'', i''), where the probability is determined by the choice ${}_p\!\oplus$. With that, we proceed with the reasoning:

$$
\begin{array}{ll}
 & (N, i) \\
\stackrel{V_x}{\mapsto} & (N{+}1, x)\ {}_{1/(N+1)}\!\oplus (N{+}1, i) \\
\stackrel{V_y}{\mapsto} & ((N{+}2, y)\ {}_{1/(N+1)}\!\oplus (N{+}2, y))\ {}_{1/N+2}\!\oplus ((N{+}2, x)\ {}_{1/(N+1)}\!\oplus (N{+}2, i)) \\
= & \hfill \text{collapse equivalent states} \\
 & (N{+}2, y)\ {}_{1/N+2}\!\oplus ((N{+}2, x)\ {}_{1/(N+1)}\!\oplus (N{+}2, i))\ .
\end{array}
$$

Overall this tells us that the initial state (N, i) is mapped to a distribution in which the final states $(N{+}2, x)$ or $(N{+}2, y)$ are both possible, each with probability $1/(N{+}2)$, and by a symmetrical argument a similar calculation for $V_y V_x$ yields the same result. The other case, where $i \in \{x, y\}$ is even simpler. Likewise a short calculation also verifies (c).

More generally, approaches for carrying out such small intricate proofs can be done using quantitative program logic described elsewhere [17]. Whilst that logic is not optimised for checking full refinements, in some special cases there

do exist practical verification rules for probabilistic refinement checking [8].

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 arithemetic calculation at al. Proofs proceed effectively by re-writing subexpressions to equivalent, or weaker ones, with the validity guaranteed by the hypotheses (in this case Fig.3) or the general pKA axiom set at Fig.2. We illustrate the style with the following calculation.

Lemma 5 *The hypotheses at Fig.3 imply (7).*

Proof: *Write* V *for* $(1{+}V_x)(1{+}V_y)(1{+}V_z)$, *and reason*

$$
\begin{array}{lll}
 & (V_x{+}V_y{+}V_z)V[T] & \\
= & V_xV[T]{+}V_yV[T]{+}V_zV[T] & \textit{Fig.2(iv) and (ix)} \\
= & V_x(1{+}V_x)(1{+}V_y)(1{+}V_z)[T] + V_yV[T]{+}V_zV[T] & \textit{Definition of } V \\
= & V_x(1{+}V_y)(1{+}V_z)[T] + V_yV[T]{+}V_zV[T] & V_x(1{+}V_x) = V_x,\ \textit{Fig.3(1)} \\
\leq & V[T]{+}V_yV[T]{+}V_zV[T] & V_x \leq (1{+}V_x);\ \textit{definition of } V \\
\leq & V[T] + V[T] + V[T] & \textit{As above and Fig.3(3)} \\
= & V[T]\ .
\end{array}
$$

From this we can appeal to Fig.2(xiii) to deduce that

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

but now $1 \leq V[T]$, *thus we have after all that the left-hand side is at least* $(V_x{+}V_y{+}V_z)^*[T]$, *and (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 Sec. 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 Sec. 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 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. To handle this, a protocol is often divided into two "phases", one for voting, and one for notification. 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 [4] to the probabilistic context, so that we may apply them to probabilistic protocol 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 Sec. 5, where we discuss proof automation.

Our first results at Lem. 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

$$a(b+1) \le ca + d \quad \Rightarrow \quad ab^* \le c^*(a + db^*) \tag{8}$$

$$ac \le cb \quad \Rightarrow \quad a^*c \le cb^* \tag{9}$$

Proof: *For (8) see Lemma 2(6) at* Computer Formalised proofs *at [5] and the appendix for (9).*

Note that the weaker commutativity condition of $ab \le ca + d$ will not do at (8), as the example with $a, b, c \hat{=} \mathit{flip}$ and $d \hat{=} \mathsf{magic}$ illustrate. In this case we see, that a^* and b^* both correspond to the program $(s := 0 \sqcap s := 1)$, and this is not the same as the corresponding interpretation for c^*a which corresponds to *flip* again.

Lem. 6 implies that with suitable commutativity between phases a and b of a program, an iteration involving the interleaving of the phases may be thought of as executing in two separated pieces. Note that again we need to use a hypothesis $b(1 + a) \le (1 + a)b$, rather than a weaker $ba \le ab$.

Lemma 7 $\quad b(1+a) \leq (1+a)b^* \quad \Rightarrow \quad (a+b)^* \leq a^*b^*$.

***Proof**: See Lemma 2 at* Computer Formalised proofs *at [5].*

Whilst the hypothesis of Lem. 7 suggests the need to examine the starred term b^*, in fact in the way we use this lemma below is by verifying the weak commutation $b(1+a) \leq (1+a)b$ — once this is done we may conclude that the more general hypothesis also holds, since $b \leq b^*$.

5 Computer support for pKA proofs

It would be a great advantage to have the possibility of verifiying pKA proofs formally with the help of a computer, as is becoming common in many of the currently used programming logics. It is possible to use for this purpose one of the several successful proof assistants to express pKA in the particular logical framework thus chosen. This choice, however, can easily turn into a major proof formalisation effort, as could be suspected by looking at the effort of Hurd in a closely related problem, the formalisation of $pGCL$ in HOL [10].

Rather than implementing the equational system from scratch, 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 [11]). 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 doesn't 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 appropiate axioms or previously proved theorems in library files. We set out the axioms and an example proof in the appendix.

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 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 [5].

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.

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). [8]

The randomised solution we consider is based on one proposed by Rabin [12]. 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 P_i. The scheduler chooses nondeterministically between the P_i, and the chosen process 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

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

not restricted in its choice, it must treat 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; the current winner's lottery number is recorded as part of the shared variable. Processes draw at most once in a competition, and the winner is notified when it executes its notification phase. The notification phase may only begin 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 [12] that in a lottery with k participants in which tickets are drawn according to (independent) exponential distributions, there is a probability of at least 1/3 of a unique winner. 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 (which led to the protocol's overall incorrectness); in fact both (a) and (b) are also not true, although the model-based argument still applies provided that the voting may be (almost) separated. We shall use an algebraic approach to do exactly that.

- *Voting phase.* P_i checks if it is eligible to vote, then draws a number randomly; 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 P_i is ineligible to vote, it skips.
- *Notification phase.* P_i checks if it is eligible to test, and if it is, then checks whether its recorded vote is the same as the maximum drawn (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.* When this 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.

---

7 The probability that a participating process loses

We now show how the lemmas of Sec. 4 can be applied to the example of Sec. 6: we show how to compute the probability that a particular process P

(one of the P_i's) participating in the lottery loses. Writing V and T for the two phases of P, respectively vote and notify (recall Fig.6) and representing scheduler choice by "+", we see that the chance that P loses can be expressed as

$$(V + T + \widetilde{V} + \widetilde{T} + C)^* \ A \ ,$$

where $\widetilde{V} \ \hat{=} \ +_{P_i \neq P} V_i$, and $\widetilde{T} \ \hat{=} \ +_{P_i \neq P} T_i$ are the two phases (overall) of the remaining processes, and A tests for whether P has lost. Thus A is a test of the form "skip if P has not drawn yet, or has lost, otherwise magic ", followed by an abstraction function which forgets the values of all variables except those required to decide whether P has lost or not.

---

(1) *Voting and notification commute*: $V_i T_j = T_j V_i$.

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

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

(4) *It's more likely to lose, the later the vote*: $VA(\widetilde{V}A + 1) \leq (\widetilde{V}A + 1)VA$.

Here V corresponds to a distinguished process P's voting phase, V_i to P_i's voting phase, and $\widetilde{V}$ to the nondeterministic choice of all the voting phases (not P's). Similarly T and T_j are the various notification phases. A essentially tests for whether P has lost or not.

Fig. 7. Algebraic properties of the system.

The crucial 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, as we did for *3-Election*. As we noted there, this task however is considerably easier than analysing an explicit model of the distributed architecture, because the code fragments can be treated one-by-one, in isolation. An alternative way to think of Fig.7 are as an *algebraic specification* for the concrete protocol code. The advantage of this approach is that our proof of correctness can be done in the algebraic style.

The next lemma uses separation to show 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.

Lemma 8 $\quad (V + T + \widetilde{V} + \widetilde{T} + C)^* \quad \leq \quad (V + \widetilde{V})^* \ C^* \ (T + \widetilde{T})^* \ .$

Proof: *We use Lem. 7 twice, first to pull $(T + \widetilde{T})$ to the right of everything else, and then to pull $(V + \widetilde{V})$ to the left of C. In detail, we can verify from Fig.7, that*

$$(T + \widetilde{T}) \ (V + \widetilde{V} + C + 1) \quad \leq \quad (V + \widetilde{V} + C + 1) \ (T + \widetilde{T})^* \ ,$$

(since $T + \widetilde{T}$ has a standard denotation, so distributes +) to deduce from Lem. 7 that $(V + T + \widetilde{V} + \widetilde{T} + C)^* \leq (V + \widetilde{V} + C)^*\ (T + \widetilde{T})^*$. Similarly

$$C(V + \widetilde{V} + 1) \quad \leq \quad (V + \widetilde{V} + 1)C^* \ ,$$

so that $(V + \widetilde{V} + C)^* \leq (V + \widetilde{V})^*\ C^*$.

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

Lemma 9 $(V + \widetilde{V})^* A \quad \leq \quad \widetilde{V}^* (VA)^*$.

Proof: *We reason as follows.*

$$\begin{array}{lll} & (V + \widetilde{V})^* A & \\ \leq & A(VA + \widetilde{V}A)^* & \textit{see below} \\ \leq & A(\widetilde{V}A)^*(VA)^* & \textit{Fig.7 (4); Lem. 7} \\ \leq & \widetilde{V}^*(VA)^* \ . & P \textit{ not voted, implies } A\widetilde{V}A = \widetilde{V} \end{array}$$

For the "see below" we note that

$$\begin{array}{lll} & (V + \widetilde{V})A(VA + \widetilde{V}A)^* & \\ = & (VA + \widetilde{V}A)(VA + \widetilde{V}A)^* A & A(V + \widetilde{V}) = (V + \widetilde{V})A, \textit{ then (9), (8)} \\ \leq & (VA + \widetilde{V}A)^*(VA + \widetilde{V}A)^* A & \\ = & A(VA + \widetilde{V}A)^* \ , & \end{array}$$

so that $(V + \widetilde{V})^* A(VA + \widetilde{V}A)^* \leq A(VA + \widetilde{V}A)^*$ by $(xiii)$, and therefore that $(V + \widetilde{V})^* A \leq A(VA + \widetilde{V}A)^*$.

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

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

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 Def. 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 $\mathcal{QS}$ [16, 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 $\mathcal{Q}S$ model is effectively the $\mathcal{L}S$ model built over this more detailed foundation.[9]

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

Hidden resolution of probability $\quad (v,h) \quad \overset{a}{\mapsto} \quad \{\,(v,\ (0\,{}_p\!\oplus 1)\,)\,\}\ .$ [10]

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

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

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.

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)

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

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

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

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

---

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

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

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 Def. 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 [11] 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 (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 with probability 1 during the voting phase, according to Def. 10 we would have

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

(because $[rn=0]+[rn=1]$ would be skip). [12] But since

$$(V+\widetilde{V}+1)([rn=0]+[rn=1]) \quad = \quad ([rn=0]+[rn=1])(V+\widetilde{V}+1)$$

we may reason otherwise:

$$\begin{array}{lll} & R(V+\widetilde{V})^*([rn=0]+[rn=1])\mathsf{chaos} & \\ = & R([rn=0]+[rn=1])(V+\widetilde{V})^*\mathsf{chaos} & \text{Lem. 6} \\ = & R[rn=0]\mathsf{chaos}+R[rn=1]\mathsf{chaos}\ , & \text{Def. 10 and (11)} \end{array}$$

which, now back in the model we can compute easily to be $\mathsf{magic}\ {}_{1/2}\!\oplus \mathsf{chaos}$, deducing that the chance that the scheduler may guess the round number is at most $1/2$, and not 1 at all.

We end by noting that this model has been worked out in detail for verifying secrecy-style properties with the probabilities abstracted [19].

[11] Here it cannot see it because it has not yet happened, not because it is hidden.
[12] Here we are abusing notation, by using program syntax directly in algebraic expressions.

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 [24]. 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 [8], 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 [21]; there are also logics for "probability-one properties" [13], and models for investigating the interaction of probability, knowledge and adversaries [6].

There are other variations on Kleene Algebra which include the relaxation of distributivity laws [18].

References

[1] KAT-ML project homepage.
`www.cs.cornell.edu/Projects/KAT/`.

[2] Kamal Aboul-Hosn. *A Proof-Theoretic Approach to Mathematical Knowledge Management*. PhD thesis, Department of Computer Science, Cornell University,

2007.

[3] Kamal Aboul-Hosn and Dexter Kozen. KAT-ML: An interactive theorem prover for kleene algebra with tests. *Journal of Applied non-Classical Logics*, 1, 2006.

[4] E. Cohen. Separation and reduction. In *Mathematics of Program Construction, 5th International Conference*, volume 1837 of *LNCS*, pages 45–59. Springer Verlag, July 2000.

[5] Carlos Gonzalia. Computer formalised pKA proofs. `www.ics.mq.edu.au/~carlos`.

[6] J. Halpern and M. Tuttle. Knowledge, probabilities and adversaries. *JACM*, 40(4):917–962, 1993.

[7] Jifeng He, K. Seidel, and A.K. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28:171–92, 1997. Available at [16, key HSM95].

[8] T. S. Hoang, C. C. Morgan, A. McIver, K. A. Robinson, and Z. D. Jin. Refinement in probabilistic b: Foundation and case study. In H. Treharne and S. Schneider, editors, *Proc. ZB 2005*, volume 3455 of *LNCS*, pages 252–273. Springer, 2005.

[9] Joe Hurd. A formal approach to probabilistic termination. In Víctor A. Carreño, César A. Muñoz, and Sofiène Tahar, editors, *15th International Conference on Theorem Proving in Higher Order Logics: TPHOLs 2002*, volume 2410 of *LNCS*, pages 230–45, Hampton, VA., August 2002. Springer Verlag. `www.cl.cam.ac.uk/~jeh1004/research/papers`.

[10] Joe Hurd, A.K. McIver, and C.C. Morgan. Probabilistic guarded commands mechanised in HOL. *Theoretical Computer Science*, pages 96–112, 2005.

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

[12] Eyal Kushilevitz and M.O. Rabin. Randomized mutual exclusion algorithms revisited. In *Proc. 11th Annual ACM Symp. on Principles of Distributed Computing*, 1992.

[13] D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53(3):165–98, 1982.

[14] A. McIver, E. Cohen, and C. Morgan. Using probabilistic kleene algebra pKA for protocol verification. In G. Struth and R. Schmidt, editors, *Proc the 9th International Conference on Relational Methods in Computer Science*, volume 4136 of *LNCS*. Springer, 2006.

[15] A. McIver and T.Weber. Towards automated proof support for probabilistic distributed systems. In *Proceedings of Logic for Programs and Automated Reasoning*, volume 3835 of *LNAI*. Springer Verlag, 2005.

[16] A.K. McIver, C.C. Morgan, J.W. Sanders, and K. Seidel. Probabilistic Systems Group: Collected reports.
`web.comlab.ox.ac.uk/oucl/research/areas/probs`.

[17] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Technical Monographs in Computer Science. Springer Verlag, New York, 2004.

[18] B. Moeller. Lazy Kleene Algebra. In Dexter Kozen and Carron Shankland, editors, *Mathematics of Program Construction*, volume 3125 of *LNCS*, pages 252–273. Springer, 2004.

[19] Carroll Morgan. *The Shadow Knows:* Refinement of ignorance in sequential programs. In *Math Prog Construction*, pages 359–378, 2006.

[20] C.C. Morgan, A.K. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–53, May 1996.
`doi.acm.org/10.1145/229542.229547`.

[21] PRISM. Probabilistic symbolic model checker.
`www.cs.bham.ac.uk/~dxp/prism`.

[22] 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.

[23] J.R. Rao. *Building on the UNITY Experience: Compositionality, Fairness and Probability in Parallelism.* PhD thesis, University of Texas at Austin, 1992.

[24] I. Saias. Proving probabilistic correctness statements: the case of Rabin's algorithm for mutual exclusion. In *Proc. 11th Annual ACM Symp. on Principles of Distributed Computing*, 1992.

[25] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, 1995.

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 commutativity 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 [5].

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 1(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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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