Using Probabilistic Kleene Algebra for Protocol Verification

AK McIver[1], E Cohen[2], and CC Morgan[3]

[1] Dept. Computer Science, Macquarie University, NSW 2109 Australia; anabel@ics.mq.edu.au

[2] Microsoft, US; Ernie.Cohen@microsoft.com

[3] School of Engineering and Computer Science, University of New South Wales, NSW 2052 Australia; carrollm@ecs.unsw.edu.au

Abstract. We describe *pKA*, a probabilistic Kleene-style algebra, based on a well known model of probabilistic/demonic computation [3, 16, 10]. Our technical aim is to express probabilistic versions of Cohen's *separation theorems*[1].

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.

Algebraic reasoning in general is very robust, and easy to check: thus an algebraic approach to probabilistic distributed systems is attractive because in that "doubly hostile" environment (probability *and* interleaving) the opportunities for subtle error abound. Especially tricky is the interaction of probability and the demonic or "adversarial" scheduling implied by concurrency.

Our case study — based on Rabin's *Mutual exclusion with bounded waiting* [6] — is one where just such problems have already occurred: the original presentation was later shown to have subtle flaws [15]. 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.

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.

One approach is probabilistic model checking, but it may quickly become overwhelmed by state-space explosion, and so verification is often possible only for small problem instances. On the other hand quantitative proof-based approaches [10, 4], 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 apply proof as a "pre-processing" stage that simplifies a distributed architecture *without the need to do any numerical calculations whatsoever*, bringing the problem within range of quantitative model-based analysis after all. It uses *reduction*, the well-known technique allowing simplification of distributed algorithms, *but applied in the probabilistic context.*

We describe a program algebra *pKA* introduced elsewhere [8] in which standard *Kleene algebra* [5] has been adapted to reflect the interaction of probabilistic assignments with nondeterminism, a typical phenomenon in distributed algorithms. Standard (i.e. non-probabilistic) Kleene algebra his been used effectively to verify some non-trivial distributed protocols [1], 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 whether that nondeterminism can "see" probabilistic choices is important [16, 3, 10]. 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.

Our case study is inspired by Rabin's solution to the mutual exclusion problem with bounded waiting [14, 6], whose original formulation was found to contain some subtle flaws [15] 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 investigate the implications on the model and algebra of adopting those assumptions which, we argue, have wider applications for secrecy and probabilities.

Our specific contributions are as follows.

1. A summary of *pKA*'s characteristics (Sec. 2), including a generalisation of Cohen's work on separation [1] for probabilistic distributed systems using *pKA* (Sec. 4);
2. Application of the general separation results to Rabin's solution to distributed mutual exclusion with bounded waiting (Sec. 5);
3. Introduction of a model which supports the algebraic characterisation of secrecy in a context of probability (Sec. 6).

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 [10] though, depending on the particular application, the conditions imposed on the subsets of (final) probability distributions can vary [12, 3]. 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. [12] 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 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*

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

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 [10]. 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 [12]). 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" [4]) 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*

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

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

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

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

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 $(\mathit{Prog} \sqcap \mathsf{skip})$ (*viz*, a finite number of compositions of $(\mathit{Prog} \sqcap \mathsf{skip})$), 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

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

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

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 $\lambda X\cdot(\mathit{Prog}; X\ \sqcap\ 1)$. All programs map $\top$ to $\{\delta_\top\}$.

Fig. 1. Mathematical operators on the space of programs [10].

where *flip* was defined at (1). It is easy to see that the iteration $\mathit{halfFlip}^*$ 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.
- *Prog* ; $[B]$ determines the *greatest probability* that *Prog* may establish B. For example if *Prog* is the program *biasFlip* at (2), then *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 of not blocking (2/3) is the maximum probability that *biasFlip* establishes $s=0$.
- Similarly, *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:*

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

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 1. *([8]) 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 *flip* ; $flip^* = s := 0 \sqcap s := 1$, i.e. *flip* ; $flip^* \neq flip$ in spite of the fact that *flip* ; $flip = flip$. 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.

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

Fig. 2. Rules of Probabilistic Kleene algebra, *pKA*[8].

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* [16]. 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. We do not deal with that stage here: indeed our purpose is precisely to make that stage a separate concern, not further complicated by the algorithm under study.

In the following sections we introduce our case study and illustrate how pKA may be used to simplify the overall analysis.

3 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). [5]

The randomised solution we consider is based on one proposed by Rabin [6]. 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 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.3. 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 [6] that in a lottery with k participants in which tickets are drawn according to (independent) exponential

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

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.4.

Fig. 3. The key events in a single round of the mutual exclusion protocol.

4 Separation theorems and their applications

In this section we extend some standard separation theorems of Cohen [1] to the probabilistic context, so that we may apply them to the mutual exclusion problem set out above. Although the lemmas are somewhat intricate we stress their generality: proved once, they can be used in many applications.

Our first results at Lem. 1 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 1.

$$a(b+1) \le ca + d \quad \Rightarrow \quad ab^* \le c^*(a + db^*) \tag{5}$$

$$ac \le cb \quad \Rightarrow \quad a^*c \le cb^* \tag{6}$$

Proof. The proof of (5) is set out elsewhere [8]. For (6) we have the following inequalities, justified by the hypothesis and monotonicity.

$$acb^* \quad \le \quad cbb^* \quad \le \quad cb^* \ .$$

Now applying (xiii), we deduce that $a^*cb^* = cb^*$, and the result now follows since $a^*c \le a^*cb^*$.

Note that the weaker commutativity condition of $ab \le ca + d$ will not do at (5), as the example with $a, b, c \mathrel{\hat{=}} \mathit{flip}$ and $d \mathrel{\hat{=}} \mathsf{magic}$ 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. 1 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 2. $\quad b(1 + a) \le (1 + a)b^* \quad \Rightarrow \quad (a + b)^* \le a^*b^*$.

Proof. *We reason as follows*

$$\begin{array}{lll}
 & (a+b)^* & \\
\le & (a+b)^*a^*b^* & 1 \le a^*b^* \\
\le & a^*b^* \ . & (a+b)a^*b^* \le a^*b^*, \textit{ see below; (xiii)}
\end{array}$$

For the "see below", we argue

$$\begin{array}{lll}
 & (a+b)a^*b^* & \\
= & aa^*b^* + ba^*b^* & \\
\le & a^*b^* + b(1+a)^*b^* & a \le a^* \le a^* \le (1+a)^* \\
\le & a^*b^* + (1+a)^*b^*b^* & \textit{hyp; } (5) \Rightarrow b(1+a)^* \le (1+a)^*b^* \\
= & a^*b^* \ . &
\end{array}$$

5 The probability that a participating process loses

We now show how the lemmas of Sec. 4 can be applied to the example of Sec. 3: 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.3) 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 $\widetilde{V} \mathrel{\hat{=}} +_{P_i \ne P} V_i$, and $\widetilde{T} \mathrel{\hat{=}} +_{P_i \ne P} T_i$ 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.

The crucial algebraic properties of the program fragments are set out at Fig.4, and as a separate analysis the verifier must ensure that the actual code fragments implementing the various phases of the protocol satisfy them. This

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. 4. Algebraic properties of the system.

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.

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

Proof. We use Lem. 2 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.4, 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. 2 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 processor P votes last, with the other processors 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 4. $(V+\widetilde{V})^*A \quad \leq \quad \widetilde{V}^*(VA)^*$.

Proof. We reason as follows.

$$\begin{array}{lll}
 & (V+\widetilde{V})^*A & \\
\leq & A(VA+\widetilde{V}A)^* & \textit{see below} \\
\leq & A(\widetilde{V}A)^*(VA)^* & \textit{Fig.4 (4); Lem. 2} \\
\leq & \widetilde{V}^*(VA)^*\ . & P \textit{ not voted, implies } A\widetilde{V}A = \widetilde{V}
\end{array}$$

For the "see below" we note that

$$\begin{array}{lll}
 & (V+\widetilde{V})A(VA+\widetilde{V}A)^* & \\
= & (VA+\widetilde{V}A)(VA+\widetilde{V}A)^*A & A(V+\widetilde{V})=(V+\widetilde{V})A,\ \textit{then (6), (5)} \\
\leq & (VA+\widetilde{V}A)^*(VA+\widetilde{V}A)^*A & \\
= & A(VA+\widetilde{V}A)^*\ , &
\end{array}$$

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. 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 $\mathcal{L}$ and pKA.

6 Secrecy and its algebraic characterisation

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

$$R(V+T+\widetilde{V}+\widetilde{T}+C)^*\ . \tag{7}$$

The problem is that the interpretation in $\mathcal{L}S$ 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).

Clearly that strategy must be prevented (if the algorithm is to be correct!) — and it is prevented *provided* the scheduler cannot see the value set by R. Thus we need a model to support algebraic characterisations for "cannot see".

The following (sketched) description of a model $\mathcal{Q}S$ [9, Key QMSRM] — necessarily more detailed than $\mathcal{Q}S$ — 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.[6]

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 $\quad (v,h) \quad \overset{a}{\mapsto} \quad \{\,(v,\ (0\ {}_p\!\oplus\ 1)\ )\,\}\ .$ [7]

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 4. *Observability; resolution.*

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

$$a(t + \neg t) \quad = \quad a\ . \tag{8}$$

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 Def. 4 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 [8] whereas, if occurring second, it could.

[6] 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 $S = V \times \overline{H}$.

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

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

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. 4 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).[9] But since $(V + \widetilde{V} + 1)([rn = 0] + [rn = 1]) \ = \ ([rn = 0] + [rn = 1])(V + \widetilde{V} + 1)$ we may reason otherwise:

$$\begin{array}{lll} & R(V + \widetilde{V})^*([rn = 0] + [rn = 1])\mathsf{chaos} & \\ = & R([rn = 0] + [rn = 1])(V + \widetilde{V})^*\mathsf{chaos} & \text{Lem. 1} \\ = & R[rn = 0]\mathsf{chaos} + R[rn = 1]\mathsf{chaos}\ , & \text{Def. 4 and (8)} \end{array}$$

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.

7 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 [15]. 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 $\mathcal{QSH}$'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 $\mathcal{QSH}$ may support a logic for probabilities and ignorance in the refinement context, though that remains an interesting topic for research.

Others have investigated instances of Rabin's algorithm using model checking [13]; there are also logics for "probability-one properties" [7], and models for investigating the interaction of probability, knowledge and adversaries [2].

There are other variations on Kleene Algebra which include the relaxation of distributivity laws [11].

A Standard equational identities that still apply in *pKA*

$$a^*a^* \quad = \quad a^* \tag{9}$$

$$a^*(b + c) \quad = \quad a^*(a^*b + a^*c) \tag{10}$$

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

References

1. E. Cohen. Separation and reduction. In *Mathematics of Program Construction, 5th International Conference*, volume 1837 of *LNCS*, pages 45–59. Springer Verlag, July 2000.
2. J. Halpern and M. Tuttle. Knowledge, probabilities and adversaries. *J. ACM*, 40(4):917–962, 1993.
3. Jifeng He, K. Seidel, and A.K. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28:171–92, 1997. Earlier appeared in Proc. FMTA '95, Warsaw, May 1995. Available at [9, key HSM95].
4. 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`.
5. D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(3):427–443, 1997.
6. Eyal Kushilevitz and M.O. Rabin. Randomized mutual exclusion algorithms revisited. In *Proc. 11th Annual ACM Symp. on Principles of Distributed Computing*, 1992.
7. D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53(3):165–98, 1982.
8. A. McIver and T. Weber. Towards automated proof support for probabilistic distributed systems. In *Proceedings of Logic for Programming and Automated Reasoning*, volume 3835 of *LNAI*, pages 534–548. Springer, 2005.
9. 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`.
10. Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Technical Monographs in Computer Science. Springer Verlag, New York, 2004.
11. B. Moeller. Lazy Kleene Algebra. In Dexter Kozen and Carron Shankland, editors, *MPC*, volume 3125 of *Lecture Notes in Computer Science*, pages 252–273. Springer, 2004.
12. 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`.
13. PRISM. Probabilistic symbolic model checker.
    `www.cs.bham.ac.uk/~dxp/prism`.
14. 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.
15. 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.
16. Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, 1995.